MASTER'S THESIS

Interuniversity Master in Statistics and Operations Research UPC-UB

Title: Creation and Development of an AI Teaching Assistant

Author: Ernest Benedito Saura

Advisor: Dr. Alexandre Perera i Lluna

Department: Department of Automatic Control (ESAII)

University: Universitat Politècnica de Catalunya

Academic year: 2017/18

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT DE BARCELONA

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master in Statistics and Operations Research
Master's Degree Thesis

Creation and Development of an AI Teaching Assistant

Ernest Benedito Saura

Advisor: Dr. Alexandre Perera i Lluna

Department of Automatic Control (ESAII)

Per a aquells qui, sense els quals, res no hauria estat com ha estat. Gràcies.

Preface

Six months before this project started I was introduced to chatbots in a 3-week Prove of Concept project. After it was finished I could not help thinking: 'I want more', and I had in mind to apply it to the educational field. When the time came that I had to choose a topic for the Master Thesis, the opportunity could not be better: a whole semester project in an educational environment. Moreover, my father, who teaches in university, was looking for new innovative ways to do teaching through the course he coordinates, which I had previously studied during the degree. Thus, I decided to do a Teaching Assistant chatbot for this subject, as I knew the content of it; I could easily clarify any doubt about it and, of course, so to help him with the teaching renovation he has been carrying out over the past years along with the rest of the course department.

Thus, this has been a multidisciplinary project, done with the collaboration of three different parts, which I would like to thank. In the first place, my advisor, Dr. Alexandre Perera, who has provided me with the guidelines of the project. Second, my father, Dr. Ernest Benedito, who has always been open to collaborate and has offered his support through all the project, both personal and professionally.

Furthermore, when this project started I had had little contact with the field of Natural Language Processing, so I got on to the university AI research department in order to get insights about it. I ended up reaching Prof. Horacio Rodriguez, who offered his experienced advice on the text processing approach I should follow. I want to acknowledge the department and specially him for his selfless implication in the project.

Finally, I want to thank all family and friends that have offered their support all along this project, with the special mention of Xavi and Sandra, who have been following the progress of it and offered their help in many occasions. Also, to Gaston, who opened the door to the chatbots world, and to many others. Thank you all.

Abstract

The interaction between a professor and a student can be difficult at a university level, due to the wide range of students that a professor has to manage or the possible concerns of a student to seem ignorant.

During this project, an end-to-end solution has been provided, creating and developing a Chatbot that, using Artificial Intelligence, is able to solve doubts that students may have within the domain of a university subject.

To do so, Advanced Statistics Techniques such as Machine Learning Classification Methods or Text Mining have been used in order to make a Natural Language Processing system that understands a question and provides either a pre-built or a generative answer, based on the documentation of the subject. This system has then been integrated into Telegram, which serves as a front-end application. Several functionalities have been incorporated, such as interactive answers, user registration, administrator rights, multiple language selection or database connections handling, among others.

Programming has been done entirely with R, both the text processing and the integration, which ended to be a reliable programming language for these kind of solutions. Moreover, the R package 'telegram', used to connect with Telegram's Bot API, has been updated in order to make possible the development of a Chatbot through Long Polling. Additionally, other user-experience enriching functionalities used in the solution have been incorporated, along with their correspondent documentation following the CRAN standards. These updates have been sent to the package author and at the end of the project are under revision so to incorporate them in the stable version of the package.

Concerning the solution provided, after the successful results of the project it is intended to initialize a Beta Phase in a real case with actual students. Hence, an innovative tool in the field of teaching has been provided, taking a step further the professor-student interaction.

Contents

List of Figures

List of Tables

List of Abbreviations

AF Answer Format. 31

AK Access Key. 48

AT Answer Type. 33

AI Artifitial Intelligence. 4

DB Database. 49

DL Deep Learning. 12

DT Decision Trees. 11

DTM Document-Term Matrix. 22

KNN k-Nearest Neighbors. 9

KW Keyword. 18

KWR KeyWordRoot. 19

LP Long Polling. 54

ML Machine Learning. 17

NB Naive Bayes. 8

NL Natural Language. 6

NLP Natural Language Processing. 6

NLU Natural Language Understanding. 6

NN Neural Network. 12

OptiSim Optimization and Simulation. 17

RF Random Forest. 11

SVM Support Vector Machine. 10

TM Text Mining. 17

UPC Universitat Politècnica de Catalunya. 17

Introductory Part

1 | Introduction

Cognition. That is the revolutionary distinctive that let humans be what we are now. For thousands of years, human's overdeveloped cognition system allowed to form complex social structures and gave the ability to share knowledge and information easily from generation to generation.

However, if someone wanted to know about a topic that was not generally known, information was harder to find. For years we have been writing books that contained such information, so one could go to a library to find it.

Nevertheless, when having doubts about a particular field that was not easily understandable, e.g. a university subject, you would probably require further explanation, so a good option was to make an appointment with an expert on that field, i.e. a professor, and clarify your concerns.

With the technological revolution, knowledge transfer took on a whole new dimension. Information was everywhere, and bibliography from many different languages could be reachable to a wide range of people. Furthermore, the boom of electronic messaging eased you to get in touch with professors or other experts of a subject.

Unfortunately, professors may not be always available; and students may have reasons not to meet them, such as scheduling issues, the desire of an immediate response or the fear to seem ignorant. But, what if we could have an expert available 24 hours per day, 365 days per year, reachable from anywhere?

Well, that is the question that motivated this whole project, leading to the creation and development of a *Chatbot*, which operates as a Teaching Assistant. As the term itself reflects, a *Chatbot* is essentially a *Chat* maintained by a ro-*bot*. In other words, a computer program that conducts a conversation via auditory or textual methods [1].

That is what has been done in this project, applied to the field of a university subject, and all along this report we will explain how.

1.1 Aim of the Project

The main goal of the project is to provide an end-to-end solution using Artificial Intelligence (AI), creating and developing a Chatbot (also refered as Bot) that is able to operate as a Teaching Assistant of a particular subject. These kind of solutions are quite infrequent in the education field, as we will see in next Chapter, so another motivation for doing this project it to open this door for possible future works.

For our use case, we have constrained the knowledge domain to Optimization and Simulation, a course taught during the sixth semester of the Bachelor's Degree in Industrial Technology Engineering at the Barcelona School of Industrial Engineering (ETSEIB) from the *Universitat Politècnica de Catalunya* (UPC), in Spain. During the report we will reference this course through its abbreviated form OptiSim or simply as 'the subject'. Additionally, you can find further information about it at its degree's Website[1].

The scope of this project has been limited to the magnitude of the Master's Thesis. Thus, we splitted this main goal into several sub-goals that we pretend to accomplish:

- Build a system that is able to **understand a potential** question of the subject, so to retrieve an appropriate answer.
- Implement the system in a **front-end application** so to make the system reachable to our target users.
- Construct a **back-end processor** which incorporates our system and assures a user-friendly experience with the front-end application.

If the project succeeds, it is expected to initialize a Beta Phase after it is delivered, for which the Chatbot will be actually tested with the student of the subject. This final part, though, is not in the scope of the project, but we will try to develop our solution contemplating this, thus adding two extra goals:

- Find a feasible way to **run the Bot continuously** with little supervision.
- Adapt the tool so that it **can be scaled** to other subjects.

As of the little precedents on this kind of solutions, it is expected that there will be a great part of autonomous learning during the project, which may vary the scope of it depending on the effort and time expended.

[1] `http://www.upc.edu/en/bachelors/industrial-technology-engineering-barcelona-etseib`

1.2 Outline of the Report

In order to structure the explanation of the project, the report was splitted into these parts:

◇ **Introductory Part**. The project is introduced, along with the state of the art.

- **Chapter 1: Introduction**. The motivation and goals of the project are presented.
- **Chapter 2: State of the Art**. We expose what was learned after the research done on state of the art Chatbots and Machine Learning Methods applied to text processing.

◇ **Part I: Text Processing**. It is explained all what concerns to the text processing, which covers from the input of a raw question to its answer.

- **Chapter 3: Text Processing Methodology**. It is explained all methodology used for the processing of text, from the problem definition to the answer generation, stating the solution approach proposed, the Text Mining and the Machine Learning techniques used.
- **Chapter 4: Experimental Framework**. We introduce the set of data available for training the system and the experiments carried out with it.
- **Chapter 5: Results Summary**. The final results of the experiment are shown.

◇ **Part II: Chatbot Implementation**. We explain the development of the front-end and back-end of the Chatbot, and how the text processing system was integrated into it.

- **Chapter 6: Design Process**. The tool specifications are introduced, along with the target user definition.
- **Chapter 7: Front-end Application**. We explain which app channel was selected for our front-end and how the Chatbot was created in it.
- **Chapter 8: Back-end Development**. The data structure of the back-end is detailed, as well as the functionalities that where implemented for the Bot.
- **Chapter 9: Moving to the Cloud**. In this final chapter of Part II it is described the server selected to maintain the Bot continuously operative. Also the overall process flow of the solution is shown.

◇ **Conclusive Part**. The conclusions of the project are exposed.

- **Chapter 10: Conclusions and Future Work**. We conclude the project evaluating the final solution presented, stating the contributions and proposing future work to do.

◇ **Appendices**. We have generated 3 documents specifying how to access the Bot, how to load its sources locally and how to scale the solution to other problems. Additionally there are 2 extra Appendixes with the auxiliar figures and R code used for the text processing.

2 | State of the Art

In this chapter we introduce the last advances on what we want to build. First, Chatbots are introduced with a brief history of them and their application to education. Second, the main Machine Learning methods that are currently used for text classifying are exposed.

2.1 Concerning Chatbots

Chatbots (abbreviation of *ChatterBot*) are typically used in dialogue systems for several practical purposes including customer service or information acquisition. Some Chatbots make use of Natural Language (NL) techniques, including Natural Language Processing (NLP) and/or Natural Language Understanding (NLU). Others only try to guess word patterns to create a response database, logically the most efficient are those that make use of NL systems [2].

2.1.1 A Brief History of Chatbots

Alan Turing theorized that if a human was talking to a truly intelligent machine, he or she would not be able to distinguish it from a human. Turing's ideas helped start the Chatbots revolution [3].

The first Chatbot, *ELIZA* was created in 1966, it could simulate the responses of a psychotherapist and could carry a conversation convincingly with a human [4]. In the 70s, ELIZA met its first non-human patient: *PARRY*, a Chatbot that imitated a person with schizophrenia and paranoia [5]. Later, in 1988, one of the first attempts to create AI through human interaction was carried out with *Jabberwacky*. It was mainly a way to entertain, with the final purpose of moving from a text-based system to one operated by voice [6].

Since then, the quick advance of technology has allowed to create much more complex AI systems, leading to one of the contemporary most famous Chatbot achievements in 2011, when IBM's *Watson*[1] won the game *Jeopardy!* against two of its best players. *Watson* was originated

[1] `https://www.ibm.com/watson/`

in 2006, and it is actually a referent in the NLP field, which uses ML to reveal features of large amounts of data [7].

During the last decade, multiple AI assistants have been born from the leading technological companies, such as Apple's *Siri*[2], Amazon's *Alexa*[3], Microsoft's *Cortana*[4] or Google's *Google Assistant*[5].

A curious episode with a Chatbot occurred on the 23^{rd} of March of 2016, when Microsoft released *Tay* via Twitter. Originally it was designed to simulate the speech and habits of a young American girl. However, as it learned from any user's inputs from the social media, *Tay* quickly developed vicious paranoia and had to be turned off in just 16 hours, since it started giving offensive comments to people [8].

2.1.2 Application to Education

In the recent years Chatbots are transforming the customer experience and, fueled by AI, they are becoming a viable customer service channel [9]. In education, though, there have not been many of these solutions, although there are two relevant cases to mention:

Jill Watson, the Teacher Assistant

IBM's AI technology is currently being used to power a teaching assistant for an online course at the Georgia Institute of Technology, named *Jill Watson* [10, 11]. After getting huge publicity, *Jill Watson* is spreading her wings and is now being implemented in universities across the globe. One of the latest to be added to the list is BI Norwegian Business School in Oslo, Norway [12].

The Deakin Genie

At the Deakin University in Victoria, Australia, development is in full swing to complete the first ever Chatbot campus genie. Just like in the case of the AI teacher assistant, the intelligence behind it comes from IBM's supercomputer system, *Watson*. Once operational, the Deakin genie will be able to answer questions related to everything a student needs to know about life on campus [13].

[2] https://www.apple.com/ios/siri/
[3] https://alexa.amazon.com/
[4] https://www.microsoft.com/en-us/windows/cortana
[5] https://assistant.google.com

2.2 Machine Learning Methods for Text Classifying

In this section we are going to introduce the Machine Learning Methods that are currently used for supervised learning classifying, applied to text processing tasks. Figure 2.1 illustrates classification problems for a set of observations (circles) and classes (colors).

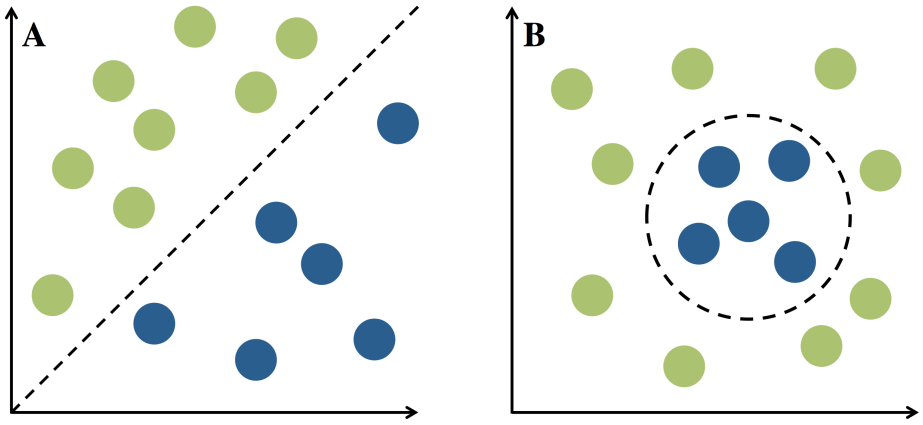

Figure 2.1: Classic Classification Problems

2.2.1 Naive Bayes

Naive Bayes (NB) is a supervised classification method for a finite number of classes c, assuming feature independence given the resulting class. It is based on the Bayes theorem, which states that being A and B two non-null events, then it is met that:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.2.1}$$

If we apply Bayes theorem to an example of supervised classifying, being d the documents that a classifier have to classify into a class c, then we have:

$$P(c|d) = \frac{P(d|c) \cdot P(c)}{P(d)} \tag{2.2.2}$$

Having a finite number of classes $c \in C$, the classifier returns the class $\hat{c}$ which has the maximum posterior probability given the document:

$$\hat{c} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c) \cdot P(c)}{P(d)} \tag{2.2.3}$$

Naive Bayes is a simple model for classification that has been applied to text classification tasks such as spam filtering [14–17], sentiment analysis [18–20] or text categorization [21, 22]. Despite its simplicity, it works quite well in text classification tasks.

2.2.2 k-Nearest Neighbors

The k-Nearest Neighbors (KNN) is a supervised classification method for a determined number of classes (it can also be used with continuous response, but it is not usual).

The idea of this algorithm is quite simple. Let $D = \{(x_1, c_1), ..., (x_n, c_n)\}$ be the set of points x_j from which the class c_j is known and $F = \{x_{n+1}, ..., x_{n'}\}$ the new observations to classify. Then, for each point to classify from $i = n + 1$ until $i = n'$:

1. Compute the distances $d_j = d(x_i, x_j)$, $j = (1, ..., n)$ for any object already classified (x_j, c_j). The habitual distance if $x_i, x_j \in \mathbb{R}^n$ is the euclidean, although any other one can be used.
2. Keep the k points from D already classified which are closest to x_i, i.e. have lowest d_j.
3. Assign to x_i the most frequent class assigned to the k selected points.
4. (Optional). Update D adding the new case, now classified.

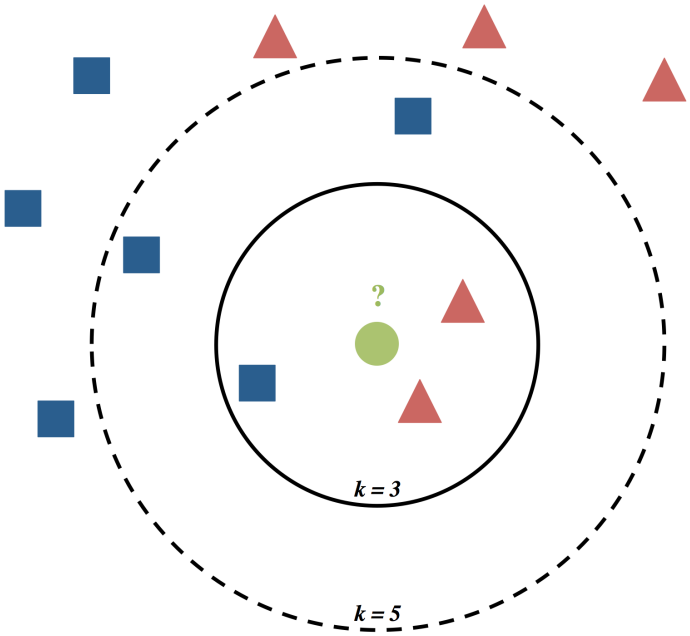

Figure 2.2: KNN Classification Example

Choosing an appropriate k is primordial for this method, as different values can give different outputs. Thas it is the case for Figure 2.2, where setting $k = 3$ or $k = 5$ would classify the new observation (in green) differently (red or blue, respectively). As it can be seen, k must be an odd number in order to discriminate in all cases.

KNN method has been used and proved to be effective for text mining purposes [23–26].

2.2.3 Support Vector Machine

Support Vector Machines (SVM) is a non-probabilistic, binary partition type of algorithm. Its foundation is that, given a training set of points from which the classes are already known, it can find an hyperplane that splits optimally the observations from each class.

Then, the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples, named *margin* within SVM's theory. Therefore, the optimal separating hyperplane maximizes the *margin* of the training data [27].

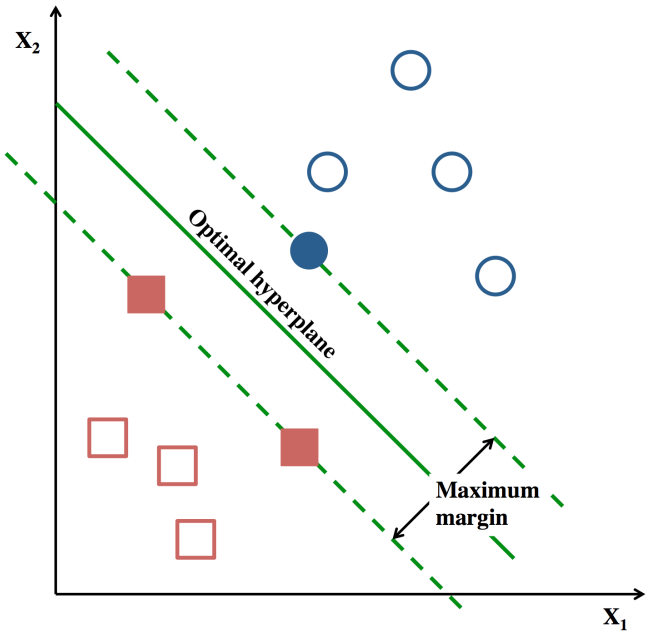

Figure 2.3: SVM Hyperplane Optimization

However, it is not always possible to separate the data within an hyperplane. To model this problem through convex optimization the *Hinge loss function* is used:

$$H = \max(0, 1 - y^{(i)}(w \cdot x^{(i)} - b)) \tag{2.2.4}$$

Where w is the vector normal to the hyperplane, the parameter b determines the offset of the hyperplane from the margin and $y^{(i)}$, $x^{(i)}$ are the class and the instance of our training dataset, respectively. We then minimize:

$$[\frac{1}{n}\sum_{i=1}^{n} H] + \lambda||w||^2 \tag{2.2.5}$$

SVMs have been used in a wide variety of NLP problems, including text classification [28–33].

2.2.4 Decision Trees

Decision Trees (DT) is a recursive partition type of algorithm. Its response can either be a qualitative or quantitative variable. It is based in the idea that, starting from a root node, data is recursively splitted based on the value of some variable that minimizes some *optimality* criteria.

There are three types of nodes:

- **Root**. Node where the tree begins.
- **Decision**. Node where the tree is bifurcated depending on the value of a variable.
- **Leaf (or terminal)**. Final node of the tree where it is stored some summarized information about the elements that belong to that leaf. If the root node is also a leaf, then the decision tree is called trivial or degenerated, case when all data is classified equally.

For classification problems, let p_{mk} be the proportion of class k observations in node m. Then, common measures of *optimality* are:

- **Gini:**
$$H(X_m) = \sum_k p_{mk}(1 - p_{mk}) \tag{2.2.6}$$
- **Cross-Entropy:**
$$H(X_m) = \sum_k p_{mk} \cdot \log(p_{mk}) \tag{2.2.7}$$
- **Misclassification:**
$$H(X_m) = 1 - \max(p_{mk}) \tag{2.2.8}$$

This method has been proved to be useful for text classification tasks [34–36].

2.2.5 Random Forest

Random Forest (RF) consists on generating a large number of DT from aleatory samples of the training set and aggregate their results. It is important, in order to avoid overfitting, to generate non-correlated trees.

This can be done trough two methods:

- Bootstrap of the observations to get the training sets.
- Aleatory sample of the variables to consider in each division.

In order to get a new observation, it is passed through all the DT and the most voted output is chosen. It can also give a probability output by averaging the outputs of the DT.

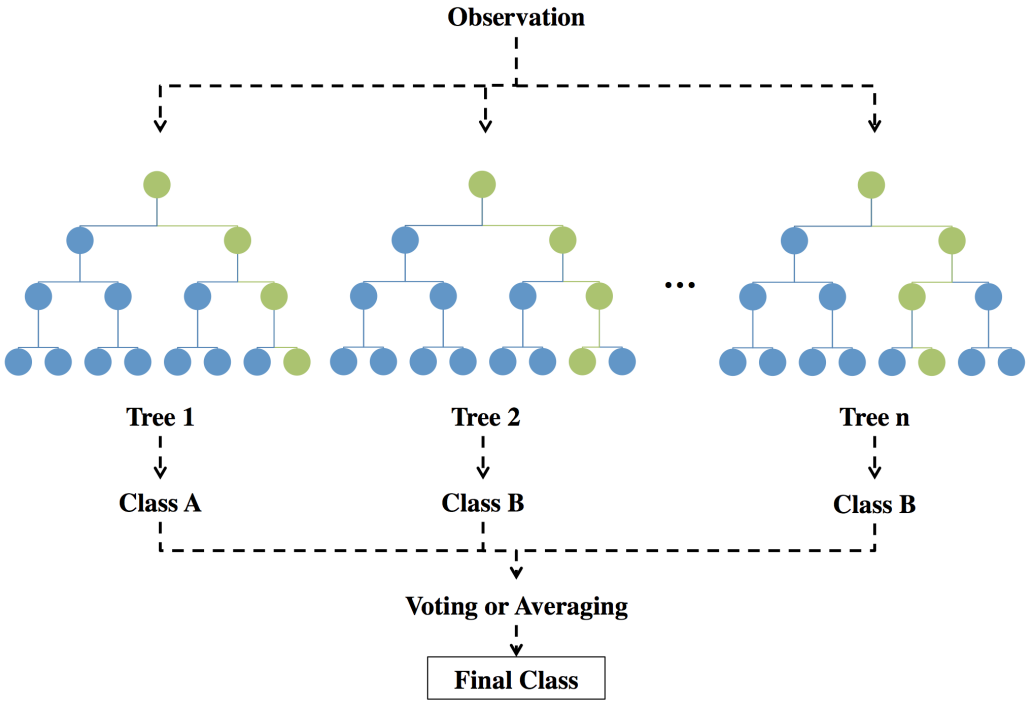

Figure 2.4: Random Forest Simplified

Since RF is an ensemble of DT that tries to avoid overfitting, it is also a good option to take into account for text classification purposes. The method has been proved to be effective for these kind of problems [37, 38].

2.2.6 Neural Networks

Neural Networks (NN) is a class of ML method inspired on the way the human brain works. Large NN architectures are also refered as Deep Learning (DL). The idea of the method is to imitate the properties observed in biological neural systems through mathematical models, and thus giving similar responses to those that the brain is capable of giving. Here we will focus on one hidden layer NN, one of the most common type of networks used in ML problems.

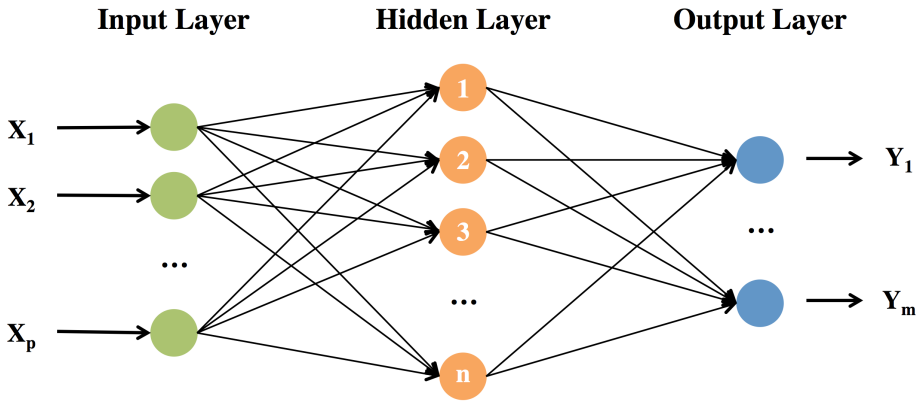

Figure 2.5: NN Structure with One Hidden Layer

A NN with one hidden layer is a non-lineal classifying method that can be represented as shown in Figure 2.5, where every node represents a neuron. A given node, namely N, has $x_1, \ldots, x_L$ input signals from all nodes of the previous layer that are connected to it. Each of these nodes are modulated by their correspondent weights $w_1, \ldots, w_L$. These inputs are composed additively with the summation $s = \sum_{l=1}^{L} x_l \cdot w_l$ and then transformed in node N by an activation function σ in order to give an output sign from such node $z_N = \sigma(s)$. Usual activation functions are:

- **Identity:**

$$\sigma(s) = s \tag{2.2.9}$$

- **Threshold:**

$$\sigma(s) = \begin{cases} 1, & \text{if } s > 0 \\ 0, & \text{if } s = 0 \\ -1, & \text{if } s < 0 \end{cases} \tag{2.2.10}$$

- **Logistic:**

$$\sigma(s) = \frac{1}{1 + e^{-s}} \tag{2.2.11}$$

- **Hyperbolic Tangent:**

$$\sigma(s) = \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}} \tag{2.2.12}$$

- **Gaussian Radial Basis Function:**

$$\sigma(s) = e^{-\frac{1}{2}s^2} \tag{2.2.13}$$

Neural Networks and derivations of it such as Convolutional NN or Recurrent NN have been applied in many fields during recent years, including NLP [39–41].

Part I

Text Processing

3 | Text Processing Methodology

In this chapter it is explained the methodology used for the text processing. This covers both the understanding of a question and the generation of an answer to it. To do so, first it is defined the problem that we are facing in this use case, in order to conceive an appropriate approach to solve it.

Once this is clear, we proceed to the explanation of the Text Mining (TM) and Machine Learning (ML) techniques used as part of the NLU. Finally we focus on the information extraction from the documents that will be sent as an answer.

3.1 Problem Definition

The main goal of the project is to design, develop and implement an *AI Teaching Assistant* restricted to the subject of Optimization and Simulation (OptiSim). The system will be integrated into a front-end app, which will be addressed in Part II. Concerning the text processing system, we can distinguish between two main blocks:

1. **Question Understanding**. This process starts with a raw question and outcomes the demand of the user, i.e. it should respond to the question *"What is the user asking for?"*. To do so, there are two things that have to be identified:

   (a) **Entities**. This is, the concept/s for which the user is asking information about. For instance, the Entity in the question *"What is an algorithm?"* would be *algorithm*. Entities depend on the domain of the system, in our case it is restricted to the OptiSim subject.

   (b) **Intent**. This is, what kind of question is the user asking. You can ask for different kind of information for the same entity. If we look at the following questions:

      i. *What is the simplex algorism?*

      ii. *How can I compute the simplex algorism?*

We can see that both questions have the same main Entity *simplex algorism*. However, in question (i) the user is asking for its definition, whilst in question (ii) he or she is looking for information about the steps of the algorism.

Thus, it is important to identify correctly both the inent and the entities of a question. In order to detect these parameters, TM and ML techniques have been used, which are explained in Section 3.3.

2. **Answer Retreive**. Once we know what the user is asking for, the second part of the problem is to provide an answer with the appropriate information to retrieve.

3.2 Solution Approach Proposed

In order to solve this problem, a tagset has been proposed to distinguish between different intents and entities. This tagset has been generated by studying questions samples of the subject (information about the questions samples dataset can be found in Section 4.1).

Intents

Thus, after making a study about the subject, four different intents have been considered for the project:

- ***DEF***. The user is asking for the definition of a term.
- ***INFO***. When asking for information of a term (how to compute, model parameters...).
- ***TIPUS***. Asking for the breakdown of a concept into its sub-parts (e.g. kind of simulation systems).
- ***YN***. When it is looking for a Yes/No answer.

Each Intent has a distinct Answer Type associated, which will be introduced in Section 3.6.2.

Entities

When revising the questions dataset, around 200 different keywords (KWs) were found. So a standardized tagset for entities has been designed. To do so, KWs were previously classified depending on the **Topic** and the **Subtopic** they belonged. These two terms will be present all along the present report, and they have been set as follows:

1. **Topic**

   The Topics of the subject were divided following the documentation of the subject. Thus, there is a Topic tag defined for each file:

   - ***TEMA1***. Chapter 1: Simulation.
   - ***TEMA2***. Chapter 2: Non-Lineal Optimization.
   - ***TEMA3***. Chapter 3: Lineal Programming.
   - ***PLE_M***. Whole Linear and Mixed Linear Programming.
   - ***PM***. Mathematical Programming.
   - ***INFORMATICS***. Corresponding to the Informatics document, where different probabilistic laws are defined.

   Additionally, three Topics that have no documentation have been added, as students recurrently have questions about them:

   - ***EXAM***. Information about the exams of the subject.
   - ***PROJECT***. The final project that students have to deliver.
   - ***TASK***. Weekly tasks from the subject.

2. **Subtopic**

   Subtopics definition have been done as well based on the structure of the subject documentation, being discussed with the coordinator of it, so to add-in his expertise on the matter. Only Subtopics for *TEMA1*, *TEMA2*, *TEMA3*, *PLE_M* and *PM* have been defined, as they are the main chapters of the subject, and thus the most extensive ones. The other Topics have been defined with a unique global Subtopic named *GLOB*. In Table 3.1 below we can find the final hierarchy of Topics and Subtopics defined.

Once the Topics and Subtopics are defined, KW tagging can be done. Each KW has a unique tag (named also ID) formed by:

$$Topic\_Subtopic\_KeyWordRoot$$

Where the KeywordRoot (KWR) is an abbreviation of the KW. To give an example, the KW ***Basic Solutions*** is from Topic ***TEMA3***, Subtopic ***PL*** and has the KWR associated ***basic_sol***. Thus, it would be tagged as:

$$TEMA3\_PL\_basic\_sol$$

This way, there is a unique ID for each KW, as there are no repeated KWRs for any particular Topic - Subtopic combination.

Topic	**Subtopic**	**Description**
TEMA1	*MODEL*	Modelization
	RELLOTGE	Clock Handling
	SIMULACIO	Simulation
	SYSTEM	Simulation Systems
	VARAL	Random Variables
TEMA2	*OSR*	Non-Restricted Optimization
	METODE	Methods
	PNL	Non-Lineal Programming
	KKT	Karush, Kuhn and Tucker Conditions
TEMA3	*PL*	Lineal Programming
	SIMPLEX	Simplex Algorism
	DUAL	Dual Systems
	QUAD	Quadratic Programming
PLE_M	*PLE_M*	Whole and Mixed Linear Programming
	LAND_DOIG	Land and Doig Algorithm
PM	*PM*	Mathematical Programming
	MODEL_PM	PM Modelization
INFORMATICS	*GLOB*	Global
EXAM	*GLOB*	Global
PROJECT	*GLOB*	Global
TASK	*GLOB*	Global

Table 3.1: Topic and Subtopic Hierarchy

3.3 Text Mining

In order to identify the intent and entities, different techniques have been used:

1. **Text Cleaning and Document-Term Matrix (DTM) Generation**. Text cleaning process, along with the DTM generation, as described in Section 3.3.1.
2. **Keywords Identification**. It concerns to the process used to identify the KWs of a question. This is detailed in Section 3.3.2.
3. **Machine Learning Classifiers**. These classifiers are built to classify the Intent, Topic and Subtopic of all questions. The accuracy of these kind of classifiers highly depends on the training dataset, features and models used. This is explained in detail in Section 3.4.
4. **Rule-based Classifier**. This is a precision-oriented classifier built by ad-hoc rules and is thought to be very precise but, since the rules are hand-crafted and the variability of questions is high, the coverage of these rules is often low.

In this Part I we will focus on the first 3 points, as the last one has only been used for questions that have been set as off-topic, i.e. questions that do not have OptiSim content (see the Section 3.6 for the off-topic definition).

When processing a message, the text cleaning and DTM generation is always done. After that, different processes are followed for the intent and entities identification:

- **Intent**. A classification has been done through a ML classifier, as it is explained in Subsection 3.5.1.
- **Entity**. The entity identification has followed a more complex process, mostly explained in Section 3.5.2, with the following layers:

Keyword Identification

+

Topic Classification

+

Subtopic Selection

+

Entity Filtering

3.3.1 Text Cleaning and Document-Term Matrix Generation

In order to start processing a question, a text cleaning has been done, proceeded by the generation of a DTM.

Text Cleaning

First, the text has been cleaned through the following steps:

1. **Lowering cases**. Upper cases have been replaced by their correspondent lower case letters.
2. **Accent remove**. Vowels accents, that may appear in Catalan and Spanish texts, have been removed. Although it is orthographically wrong, accents are often not written in those languages, specially when chatting. If we didn't remove the accents, the same word written with or without accent would be treated as different words. Thus, removing them provides a more robust analysis.
3. **Punctuation replace**. Apostrophes have been replaced by whitespaces, so to separate different words that are linked by an apostrophe. All other punctuation characters have been removed.
4. **Number remove**. Numbers have been removed, as they are not needed in this use case.
5. **Strip white space**. Extra whitespaces from the text have been also removed.

These steps have been determined by comparing which combination of them outcomed a higher accuracy of classification.

Document-Term Matrix Generation

A Document-Term Matrix (DTM) is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a DTM, rows correspond to documents in the collection and columns correspond to terms [42].

In order to generate the DTM, we have treated each question as a different document. Thus, to give an example, if we did the DTM of the sentence:

What is an algorithm?

Along with its Spanish translation:

¿Qué es un algoritmo?

	algorithm	an	is	what	algoritmo	es	un	que
L1	1	1	1	1	0	0	0	0
L2	0	0	0	0	1	1	1	1

Table 3.2: DTM Example

After applying the text cleaning methods described previously we would get the DTM presented in Table 3.2. As we can see, each line corresponds to a question; and column variables correspond to the distinct terms that appear in those questions. The matrix is then filled with the corresponding frequency of a term in a particular question.

DTM are very useful in the field of NLP, and we will see its usability in our case when modeling the ML classifiers in Section 3.4 and for the document scrapping in Section 3.6.1.

3.3.2 Keywords Detection

First though, we will explain how the KWs have been identified, as they will appear as well in the ML classifiers modeling.

To do so, we have previously generated a list with the KW found in our particular domain. We can see in Table 3.3 a sample of it, with the following columns containing information about each KW:

- **Topic**. Topic that the KW belongs to.
- **Subtopic**. Subtopic that the KW belongs to.
- **Keyword**. KeyWordRoot of the KW (explained previously in Section 3.2).
- **ID**. Unique identifier of the KW (also introduced in Section 3.2).
- **Parents**. A hierarchy of keywords have been generated, so in this column we can find the other keywords from which the actual KW is derived.
- **n**. Number of words of the KW. It can vary from 1 to 4.
- **unique**. Binary variable indicating whether the KW is particular for that Topic (i.e. it does not appear in other Topics) or not.
- **Languages**. One column for each language available in the tool is generated, containing ways of saying a KW in the different languages (see available languages in Section 6.1.3).

	Topic	Subtopic	Keyword	ID	Parents
L1	TEMA1	MODEL	continu	TEMA1_MODEL_continu	TEMA1_MODEL_model
L2	TEMA1	MODEL	determinista	TEMA1_MODEL_determinista	TEMA1_MODEL_model
L3	TEMA1	MODEL	dinamic	TEMA1_MODEL_dinamic	TEMA1_MODEL_model
L4	TEMA1	MODEL	discret	TEMA1_MODEL_discret	TEMA1_MODEL_model
L5	TEMA1	MODEL	esdev	TEMA1_MODEL_esdev	
L6	TEMA1	MODEL	estatic	TEMA1_MODEL_estatic	TEMA1_MODEL_model
L7	TEMA1	MODEL	estocastic	TEMA1_MODEL_estocastic	TEMA1_MODEL_model
L8	TEMA1	MODEL	model	TEMA1_MODEL_model	
L9	TEMA1	MODEL	var_est	TEMA1_MODEL_var_est	TEMA1_MODEL_model
L10	TEMA1	RELLOTGE	asincron	TEMA1_RELLOTGE_asincron	TEMA1_RELLOTGE_gestio_rel
L11	TEMA1	RELLOTGE	gestio_rel	TEMA1_RELLOTGE_gestio_rel	
L12	TEMA1	RELLOTGE	sincron	TEMA1_RELLOTGE_sincron	TEMA1_RELLOTGE_gestio_rel

	n	unique	Catalan	Spanish
L1	1	0	continu, continua	continuo, continua
L2	1	0	determinista, deterministes	determinista, deterministas
L3	1	0	dinamic, dinamica	dinamico
L4	1	0	discret, discrets, discreta, discretes	discreto, discretos, discreta, discretas
L5	1	0	esdeveniment, esdeveniments, llista	evento, eventos, lista
L6	1	0	estatic, estatics, estatica, estatiques	estatico, estatica, estaticos, estaticas
L7	1	0	estocastic, estocastica, estocastics, estocastiques	estocastico, estocastica, estocasticos, estocasticas
L8	1	0	model, models	modelo, modelos
L9	2	1	variable, variables, estat, estats	variable, variables, estado, estados
L10	1	1	asincron, asincrona, asincrons, asincrones	asincrono, asincrona, asincronos, asincronas
L11	2	1	gestio, rellotge, gestions, rellotges	gestion, reloj, gestiones, relojes
L12	1	1	sincron, sincrons, sincrona, sincrones	sincrono, sincrona, sincronos, sincronas

Table 3.3: Keywords Sample

Thus, once the language of the text is detected (this will be seen in detail in Part II), we can retrieve the appropriate column of language from Table 3.3 and cross it with the terms of the text. We can say that a KW has been identified when a possible combination of the n words of that KW are found. To give an example with the KWs from table 3.3, for the sentence:

¿Qué variables necesito para la gestión de reloj asíncrona?

The system would get the Spanish column words, and find the KWs with IDs:

TEMA1_RELLOTGE_gestio_rel and ***TEMA1_RELLOTGE_asincron***

The KW with ID *TEMA1_MODEL_var_est* would not be detected, as it is a 2-word KW, so it would need the additional word *estado* in order to be identified as a KW.

3.4 Machine Learning Classification Models

Now that we have clarified the pre-process of the text, in this section we are going to focus on the classification models that have been built. We needed to classify 3 classes: **Intent**, **Topic** and **Subtopic**; for 2 languages, **Catalan** and **Spanish**. So 6 different models were built.

	Catalan	**Spanish**
Intent	$Model_1$	$Model_2$
Topic	$Model_3$	$Model_4$
Subtopic	$Model_5$	$Model_6$

Table 3.4: Schema of Models Built

The classification outputs of those models are then used for the identification of the Intent and the Entities of a question. In order to build them, we followed an iterative methodology, explained below. Detailed explanation of the experiments carried out with this methodology can be found in Section 4.2 from next Chapter.

Furthermore, it is important to remark that we used R for the ML methods, a state-of-the art programming language and one of the most popular ML tool used by data scientist and in academics, because of the breadth of techniques it offers and its open source software [43].

3.4.1 ML Methods Comparison

As we've seen in Section 2.2, there are many ML methods available for NLP. In order to choose the best classification method for our case, a comparison analysis has been done with little tuning, so to make a preliminary filtering. The methods compared where:

- **Naive Bayes**. Using `naiveBayes` function from `e1071` package and trying:
  - $l = 1, \ldots, 10$, where l is the Laplace smoothing.
- **k-Nearest Neighbors**. Using `knn` function from `class` package and trying:
  - $k = 3, 5, \ldots, 31$, being k the number of neighbors considered.

- **Support Vector Machine**. Using `svm` function from `e1071` package. Setting the Kernel factor $e^{-\gamma \cdot |u-v|^2}$ and trying:
  - $\gamma = 2^n, \quad n = -15, -14, \ldots, 3.$
- **Decision Tree**. Using `rpart` function from `rpart` package.
- **Random Forest**. Using `randomForest` function from `randomForest` package, training with 200 trees and trying:
  - $mtry = 2^n, \quad n = 1, \ldots, 8$, where $mtry$ is the number of variables randomly sampled as candidates at each split.
- **Neural Networks**. Using `nnet` function from `nnet` package, with one hidden layer and trying:
  - $size = 1, \ldots, 5$, where $size$ is the number of units in the hidden layer.
  - $decay = e^{\lambda} - 1, \quad \lambda = 0, 0.5, 1, \ldots, 4$, being $decay$ the weight decay parameter.

In Table 3.5 we can find the performance summary of the different models, picking the set of parameters which gave the highest accuracy for each of them. The results are for language Catalan, although for Spanish they are almost identical, so only Catalan is displayed.

	Intent Acc. (%)	Topic Acc. (%)	Subtopic Acc. (%)
NB	9.8	4.9	34.1
KNN	78.0	68.3	61.0
SVM	81.7	75.6	65.9
DT	76.8	64.6	56.1
RF	**90.7**	**82.2**	**80.2**
NN	86.6	80.5	11.3

Table 3.5: ML Methods Performance Summary

After the analysis, **Random Forest has been selected** as the ML method used for classification for all models. Nonetheless, we considered also the using of NN, as building NN with several layers could have improved considerably the accuracy. However, as the RF performance was high enough for our solution, we decided to leave large/deep NNs architectures for future works. The code used for this analysis can be found in Appendix E.

3.4.2 Select Predictor Variables

For each model, we had to select an appropriate matrix of predictors **X** (i.e. the matrix with the significant variables for prediction), in order to predict **Y**, the response vector. We finally used:

- **Intent Classifier**
  - **X: DTM**. We created a DTM of the questions dataset for each language, removing sparse words columns. This left us with a total of 362 variables (i.e. distinct words) for Catalan and 360 for Spanish.
  - **Y: Intent Class**.
- **Topic Classifier**
  - **X: DTM** + ***unique*** **frequency**. In Section 3.3.2 we saw that there were some KW which were particular from a Topic, i.e. they did not appear in other Topics. For the Topic Classifier we used as predictor variables the DTMs used for the Intent Classifier plus the amount of *unique* words from each Topic present in the questions, creating thus one extra column per Topic. The addition of these 9 variables increased the performance of the models.
  - **Y: Topic Class**.
- **Subtopic Classifier**
  - **X: DTM** + ***unique*** **frequency** + **Topic**. For the Subtopic Classifier we used the variables from the Topic Classifier plus an extra variable that contained the Topic of each question. For the testing phase we filled this column with the Topic predicted by the Topic Classifier.
  - **Y: Subtopic Class**.

For each language we used the same structure of variables.

3.4.3 Data Partition

In order to check the performance of a model, we used a train-test methodology, being:

a) ***Training Set***. A fraction of the entire dataset was used for training purpose.

b) ***Testing Set***. The model trained with the *Training Set* was tested with the the *Testing Set*. Thereby, we checked the model performance with observations that were not used for the training of it.

Furthermore, for the performance computation of a model, we did 20 different folds, i.e. 20 random sampling in order to get 20 training and test sets, with a proportion of 70 % train and 30 % test. Thereby, the performance of a model has been computed by:

1. **Training** 20 models $RF_1(args, X_1), \ldots, RF_{20}(args, X_{20})$, where $args$ are the fixed arguments we are checking and $X_1, \ldots, X_{20}$ the 20 training sets folds.
2. **Predicting** the 20 test set folds with the correspondent models and compute the percentage of classifying accuracy.
3. **Averaging** the 20 classifying accuracies, giving thus the overall performance value.

3.4.4 Tune the RF Parameters

The main parameter to tune in a RF is the number of variables randomly sampled as candidates at each split, namely *mtry*. This is, when forming each split in a RF tree, the algorithm randomly selects *mtry* variables from the set of predictors available. Hence, when forming each split a different random set of variables is selected within which the best split point is chosen. Thus, in order to tune the *mtry* parameter we did:

- 1^{st} **Iteration**

  A first iteration trying a set of *mtry* is done. To do so, we choose the *mtry* values $2^1, 2^2, \ldots, 2^N$ such that $2^N \leq v < 2^{N+1}$, where v is the number of variables of the model. We then check the performance of each value.

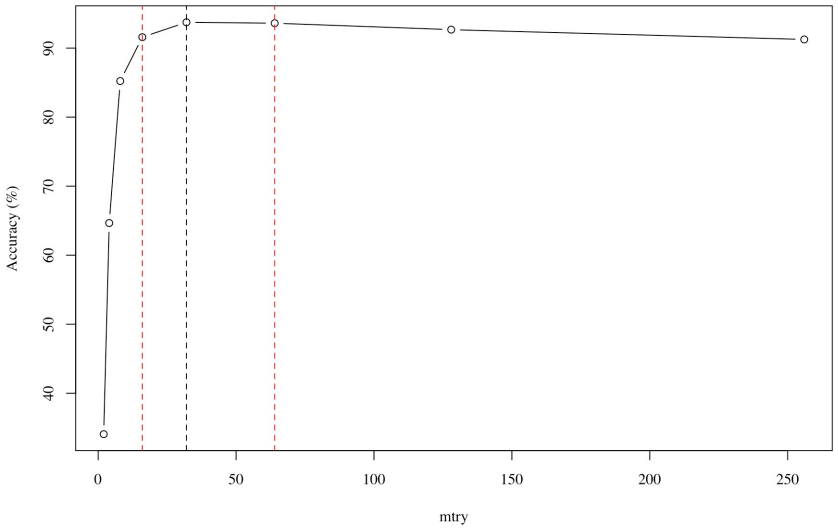

Figure 3.1: Example of 1st iteration for Spanish Topic Classifier

In Figure 3.1 we can see an example of 1^{st} iteration. The black vertical line indicates the optimum, while the red lines indicate the $[T_{i_L}, T_{i_U}]$ range of the next iteration, which is explained next.

- i^{th} **Iteration**

  For a given iteration $i > 1$, let $m_{i_1}, \ldots, m_{i_n}$ be all the *mtry* values tried during the $i-1$ previous iterations to i; ordered in an ascendant order, i.e. $m_{i_1} < \cdots < m_{i_n}$.

  Let $p_{i_1}, \ldots, p_{i_n}$ be the correspondent performances of the models trained with such *mtry* values. And let m_{i_k} be the optimum *mtry* value such that $p_{i_k} = \max(p_{i_1}, \ldots, p_{i_n})$.

  Hence, at the i^{th} iteration we will choose a set of *mtry* values to check $m'_{i_1}, \ldots, m'_{i_{10}}$ within the range $[T_{i_L}, T_{i_U}]$, where $T_{i_L} = m_{i_{k-1}}$ and $T_{i_U} = m_{1_{k+1}}$. If $k = 1$ or $k = n$: if $i = 2$, $T_{i_L} = 1$ or $T_{i_U} = v$, respectively; for $i > 2$, $T_{i_L} = m_{1_k}$ or $T_{i_U} = m_{1_k}$, respectively.

  Within this range, we will get 10 values computed as:

$$m'_{i_j} = [T_{i_L} + j \cdot \frac{T_{i_U} - T_{i_L}}{11}], \qquad j = 1, \ldots, 10 \tag{3.4.1}$$

  We will then check the model performance for each *mtry* value $m'_{i_1}, \ldots, m'_{i_{10}}$ that has not been checked before, i.e. it is not contained in $m_{i_1}, \ldots, m_{i_n}$.

- **Final Iteration**. These iterative process continues until there is an iteration for which all the values $m'_{i_1}, \ldots, m'_{i_{10}}$ were already tried in previous iterations.

Once the *mtry* is optimized, we have to choose a large enough number of trees *ntree* for the RF so that the error of predictions converges.

3.4.5 Error Analysis and Re-adjustment

Next, a document with the mistaken predictions is generated, in order to analyze the causes of the errors. Iteratively, a revision of the model is done depending on the conclusions made from the analysis. During this analysis we decided to set a predicting probability *threshold* for which we will consider valid the classification.

3.4.6 Model Performance

Finally, when the *mtry* and *ntree* arguments are optimized, we generate q RF models with such parameters and training data $X_1, \ldots, X_q$, where q is the number of questions in the whole dataset, and $X_1, \ldots, X_q$ the subset of the dataset which contains all questions except for question q. This way we can check the performance for each question for the optimized values.

3.5 Intent and Entity Identification

Once the models are built, we have identified the Intent and Entities of new questions following the methodology described below.

3.5.1 Intent Detection

This has been set directly from the Intent Classifier output, setting a *threshold*. Predictions below that *threshold* have been set as *INFO*, as that is the broader Intent. The Intent detected will determine the Answer Type of the retrieved answer (defined below in Section 3.6.2).

3.5.2 Entity Identification

For the Entity identification, four steps have been done:

1. **Keyword Detection**. We first get all KWs detected in the text as described previously in Section 3.3.2.

2. **Topic Classification**. Second, we get the output from the Topic Classifier, selecting the highest probability Topic class predicted.

3. **Subtopic Selection**. Once we've predicted the Topic, we get a Subtopic prediction from the Subtopic Classifier. More than one Subtopic can be set as candidate for the question being evaluated, selected through the following process:

   Given $S_1, ..., S_N$ the Subtopics from the Topic classified, the Subtopic Classifier returns the outputs $p_1, ..., p_N$, being p_n the probability weights of each Subtopic, meeting that $p_n \in [0, 1], \quad n = 1, ..., N$ and $\sum_{n=1}^{N} p_n = 1$.

   Therefore, we ordered $p_1, \ldots, p_n$ in a descendant order, i.e. $1 \geq p_{i_1} \geq \cdots \geq p_{i_n} \geq 0$ and selected the minimum number of Subtopics $S_{i_1}, \ldots, S_{i_K}$ such that $\sum_{k=1}^{K} p_{i_k} > threshold$, where $threshold \in [0, 1]$. In other words, we selected the first K Subtopics such that the sum of their weights was beyond a threshold.

4. **Entity Filtering**. Finally we have filtered the initial keywords detected in step 1. This filtering has been done by:

   (a) **Wrong Topic/Subtopic Removal**. KWs that are not from the Topic classified or the Suptopic selected are removed.

   (b) **Parents Removal**. For each of the remaining KWs, its parents have been removed.

3.6 Retrieval Based Answer Generation

Once explained the text processing methodology, in this section we are going to introduce how the answer retrieval is done. However, we must say that, as we will see in next Chapter, the information retrieval will only be applicable if some conditions are met from the classifiers outputs. Questions that do not meet these requirements will be treated differently. Thus, the filters done are the following:

1. **Topic Classifier output beyond a threshold**. When the Topic Classifier is applied, the output is a vector of the different Topics and a weight for each of them between 0 and 1, summing 1. In order to select a Topic, we take the Topic with the highest weight. However, if the maximum weight is below a threshold, we have determined that question is not about our domain, i.e. it has no content from the subject OptiSim. In our case this threshold is 0.5 (see Section 4.2 for the threshold selection experiments).

2. **One and only one Entity is detected**. If the question meets the first requirement, then the Subtopic Classifier and Entity Filtering is done. However, we will only provide information if one and only one Entity remains at this point. The other cases (whether there are no Entities or there are more than one) are treated through other Answer Formats.

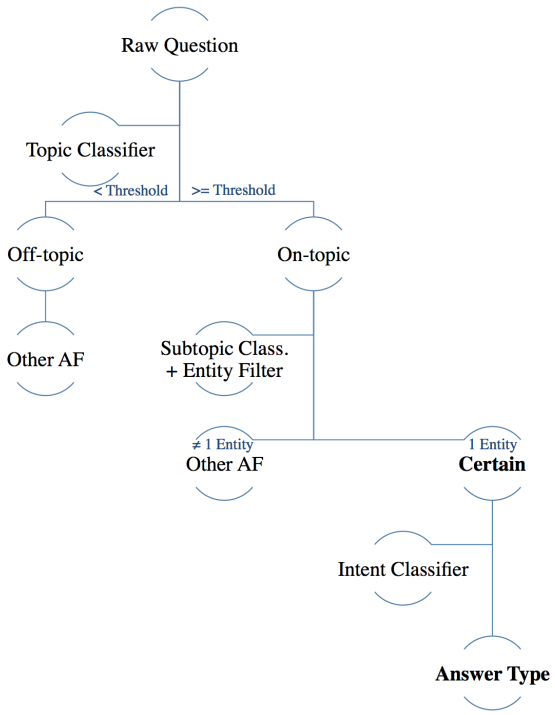

Figure 3.2: Text Processing Decision Tree

In Figure 3.2 we can see this decision process summarized. Additionally, we must differentiate between two answer characteristics:

- **Answer Type (AT)**. Defined by the Intent.
- **Answer Format (AF)**. Depending on the output of the classifiers, different AF are retrieved. In this Part we are going to focus only on the AF **Certain**, as it is the only one that requires an AT approach.

Regarding to the other AFs, they are treated later in the Chatbot Implentation. You can find further information about them in the Answer Formats Section (Section 8.2) in Part II.

3.6.1 Document Scrapping and Information Extraction

In order to scrap the documents and extract information from them, we used the following methodology:

1. **Select a File**

   Let $F_1, \ldots, F_6$ be the `pdf` files from the subject documentation.

   As described previously, each file has associated a Topic class. Therefore, being T_c the Topic class from the output of the Topic Classifier; f T_c has a file F_c associated to it, we will make the document scrapping with it; otherwise default answers will be retrieved.

2. **Generate File DTM**

   Let N and M be, respectively, the number of pages and the number of distinct words of the file F_c. We generate a DTM, namely D, of the file F_c, with N rows and M columns, treating each page of F_c as a distinct document. Thereby, the value D_{nm} is the frequency of the word corresponding to the column m in the page n.

3. **Intersect Question Words**

   Let $C_1, \ldots, C_K$ be the columns from the DTM D, whose corresponding words are present in the question being evaluated, Q.

4. **Return Question Words Document Frequency**

   Finally, we return the frequencies of the filtered words present in each page of the file $f_1, \ldots, f_N$, computed as:

$$f_n = \sum_{k=1}^{K} D_{n,C_k}, \qquad n = 1, \ldots, N \tag{3.6.1}$$

3.6.2 Answer Types

In Section 3.2 we introduced that each intent has an Answer Type (AT) associated. In Table 3.6 we can find the Intent - AT correspondence.

Intent	**AT**	**Description**
DEF	*TEXT*	Definition Text
INFO	*DOC*	Fragment of File
TIPUS	*LIST*	List of Entities
YN	*BOOL*	Yes or No

Table 3.6: List of Intents and AT associated

The methodology used for generating the answers to each AT is described below.

TEXT

For a given Entity, when the Intent detected is *DEF* it means the user is asking for a definition of a concept. In this case, we used an hybrid methodology depending on the performance through the different Entities.

On the one hand, we used the document scrapping methodology explained previously. Being $f = (f_1, \ldots, f_N)$ the frequencies of the filtered words present in each page of the file, we retrieve the text from the page with frequency $f_n = \max f$, i.e. we retrieve the text inside the page with the highest frequency.

In the other hand, we implemented a method for which the definitions are manually written into a DB, this can improve the correctness of the answer and allows to send a response in the different languages, as now with the first method the answers are in the language of the documentation files of the subject, which is Catalan.

DOC

Moreover, when the *INFO* Intent is detected, the system sends the file where the Entity information is located, and tells the user which pages to look inside the file. To do so, we used the document scrapping methodology explained previously, and retrieved the pages whose frequency of words are beyond a threshold $th \in [0, 1]$, defined as a percentage of the maximum

page frequency, i.e. we retrieve the pages with frequencies $f_n \geq \max f \cdot th$. For this case we used $th = 0.8$.

LIST

For the case of a *TIPUS* Intent, the system retrieves the breakdown of a particular Entity, listing the Entities that depend on it. In Figure 3.3 we can see an example of a *LIST* AT.

Figure 3.3: Example of a *LIST* AT

BOOL

This AT was defined in the original problem definition but was accorded to be removed as the project developed, as it needed the detection of interaction between Entities. We decided to re-arrange questions classified with this AT to *DOC*, leaving the development of this AT for future work.

4 | Experimental Framework

Once explained the methodology used for our problem, we will now explain the set of data available and the experiments carried out with it.

4.1 The Dataset

The dataset used for the experiments was kindly provided by the coordinator of the subject in scope. Thus, he provided a list with 232 common questions that the students may have, written in Catalan. In order to prepare the data, we did the following:

1. **Create Spreadsheet**. We created a spreadsheet with one column, named *catalan*, containing one question per row.
2. **Translation**. The questions were translated to Spanish so to make also models in that language. Thus, a second column *spanish* was generated.
3. **Intent Selection**. As the methods used are supervised, we needed to classify beforehand the training dataset, creating thus three additional columns in the spreadsheet. After the problem definition was done, we set an Intent to each question through a new variable.
4. **Topic Tagging**. A variable indicating the Topic of the questions was generated
5. **Subtopic Tagging**. Also the Subtopic variable was done.
6. **Keyword Hierarchy Generation**. Finally, the hierarchy of KWs was generated, identifying a total of 211 KWs.

With this done, we could proceed to do the building of the models.

4.2 Experiments

Below we will show the experiments carried out through the methodology introduced in the previous Chapter, in Section 3.4, in order to tune the model parameters.

4.2.1 Optimum *mtry*

Following the methodology described previously we proceeded to calculate the optimum *mtry*, obtaining the values shown in Table 4.1, along with the correspondent accuracies.

	Catalan		**Spanish**	
Classifier	**mtry**	**Acc. (%)**	**mtry**	**Acc. (%)**
Intent	44	91.4	181	91.6
Topic	34	96.3	32	93.7
Subtopic	195	93.8	80	91.8

Table 4.1: Optimum mtry and Accuracy for each model

In Figure 4.1 we can see the optimization process applied to the Topic Classifier for Spanish language. We can see the average performances of the different *mtry* values checked in black. The vertical line indicates the optimum *mtry* value, which in this case is 32, giving an accuracy of 93.7 %. Correspondent Figures of the other models are located in Appendix D, Figures D.1-D.5.

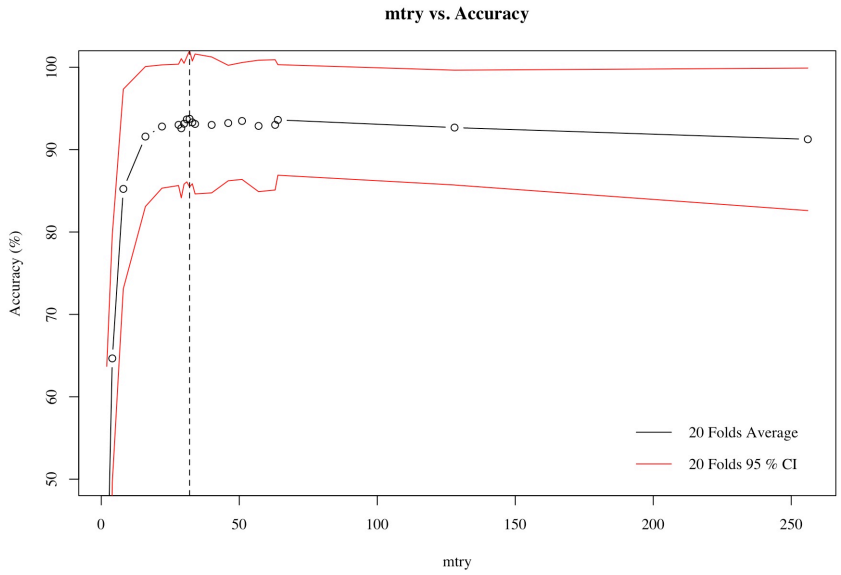

Figure 4.1: Optimum mtry for Spanish Topic Classifier

4.2.2 Converged *ntree*

Once optimized the *mtry* we had to find an appropriate *ntree* that assured the convergence of the errors. To do so, we trained the model with $ntree = 1, \ldots, 250$ looked the one with the highest accuracy.

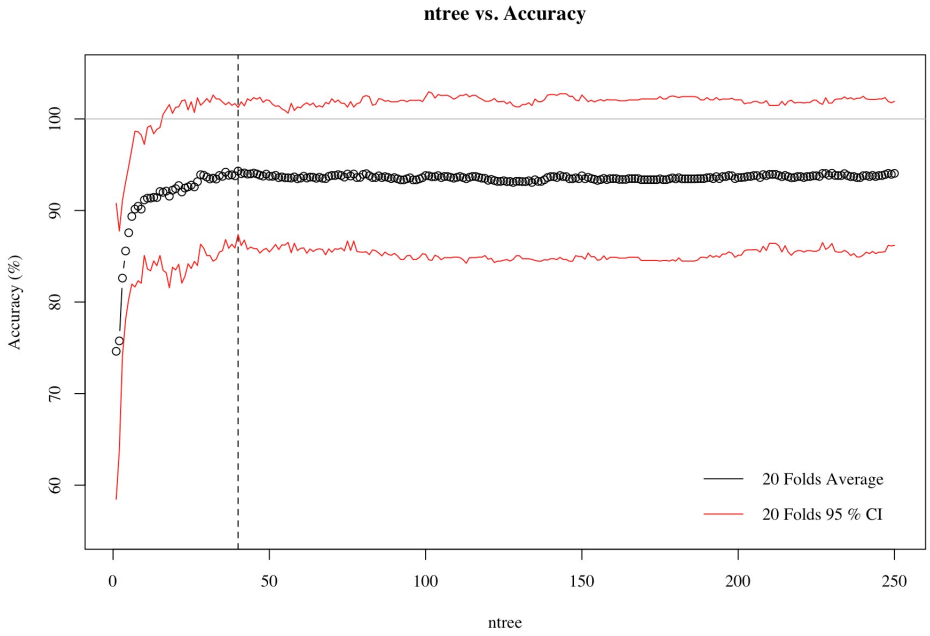

Figure 4.2: Converged ntree for Spanish Topic Classifier

By taking this value we assure that the error has converged. In Figure 4.2 we can see the convergence for the Spanish Topic Classifer. The correspondent Figures of the other models can be found in Appendix D, from Figure D.6 to D.10.

In table 4.2 we can find the *ntree* value for each model, along with the correspondent accuracies.

	Catalan		**Spanish**	
Classifier	**ntree**	**Acc. (%)**	**ntree**	**Acc. (%)**
Intent	43	91.6	202	92.1
Topic	64	96.6	40	94.3
Subtopic	86	94.6	50	91.8

Table 4.2: Converged ntree and Accuracy for each model

4.2.3 Error Analysis

Having the parameters tuned, we proceeded to compute the final performance of the model. As explained before in Section 3.4.6, to do so we generated q RF models with such parameters and training data $X_1, \ldots, X_q$, where q is the number of questions in the whole dataset, and $X_1, \ldots, X_q$ the subset of the dataset which contains all questions except for question q. Checking then the performance for each question.

Next, made an analysis of the misclasifications. In the first iterations, we found out some of the misclasified questions had been wrongly tagged, which led us to re-tag them and re-train the models.

Once the errors were corrected, we could get the final predicting accuracies shown in Table 4.3.

	Catalan	**Spanish**
Intent Classfier	91.4	93.1
Topic Classfier	97.0	94.0
Subtopic Classfier	91.4	88.8

Table 4.3: Models Performance (%)

At this point, we analyzed again the misclasified and observed that most of them where broad questions with words from different Topics. Thus, we focused on the predicting probability output of each model and set a minimum *threshold = 0.5*. Predictions with probabilities beyond this *threshold* were considered not reliable. As we saw in last Chapter, this have the following implications for each class:

- **Intent**. Intents beyond this *threshold* were set to the default class *INFO*, as it is the broader one.
- **Topic**. We set the question as off-topic, which requires different treatment, explained in Part II.
- **Subtopic**. We considered as plausible Subtopics the first K number of them such that the sum of their weights were above the *threshold*.

For the *threshold* selection, we grouped the questions by percentiles, depending on the predicted probability. Thereby, we were able to compute the prediction accuracy for each percentile. In Figure 4.3 we can see, on the left, the amount of claims of each percentile for the Spanish Suptopic Classifier. On the right it is shown the predictions accuracy average for such percentile. As it can be seen, predictions with predicted probabilities higher than the *threshold* are usually well classified. As all models gave similar results, we decided to take a global *threshold* of 0.5.

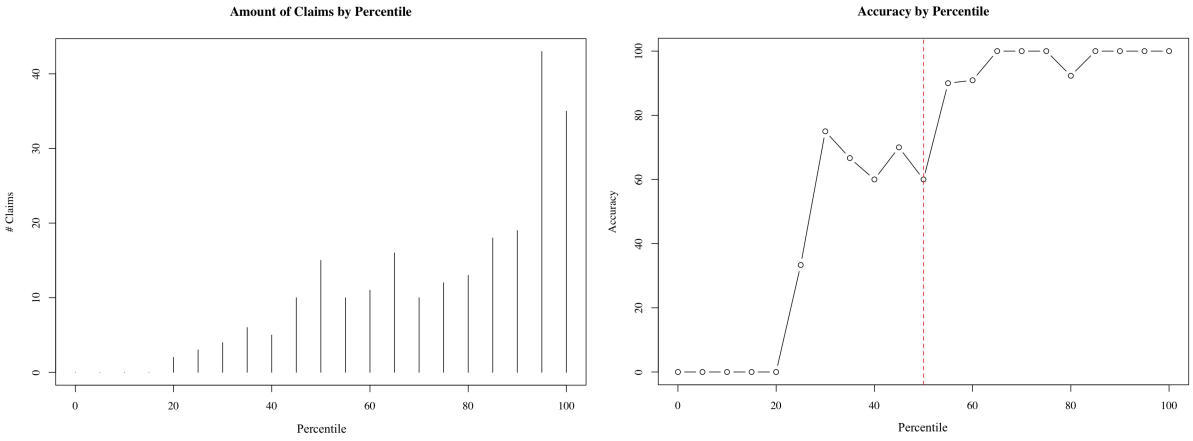

Figure 4.3: Amount of Claims and Accuracy Average per Percentile for Spanish Subtopic

Of course, by setting this *threshold* we improve our accuracy, but reject some questions, in Table 4.4 we can see the percentage of improvement and rejection for each model.

	Catalan		**Spanish**	
Classifier	**Improved**	**Rejected**	**Improved**	**Rejected**
Intent	0.5	9.5	1.1	8.7
Topic	1.2	12.9	2.5	7.3
Subtopic	4.6	19.2	5.9	17.7

Table 4.4: Threshold Application Improvement and Rejection (%)

4.2.4 Code Adaptation and Optimization

Finally, we had to adapt the code and models to the solution, so we generated a function that, given a question, its language, and the class wanted to predict, it would return:

- **Intent** predicted with its predicted probability.
- **Topic** predicted with its predicted probability.
- **Subtopics** selected.

During the generation of this function, we optimized the code so to make the processing of each question faster. To do so, we:

- Used `data.table` objects from `data.table` package, which makes the data management and computations faster.
- Moved out of the function all variables that could be loaded previously.
- Avoiding loops and using R in-build functions such as `apply`, among others, when possible.

This changes allowed us to process the 232 training questions at an average speed of `62.5 ms` per question for Catalan language and `63.4 ms` for Spanish. In Figure 4.4 we can see the timing distributions of such processing.

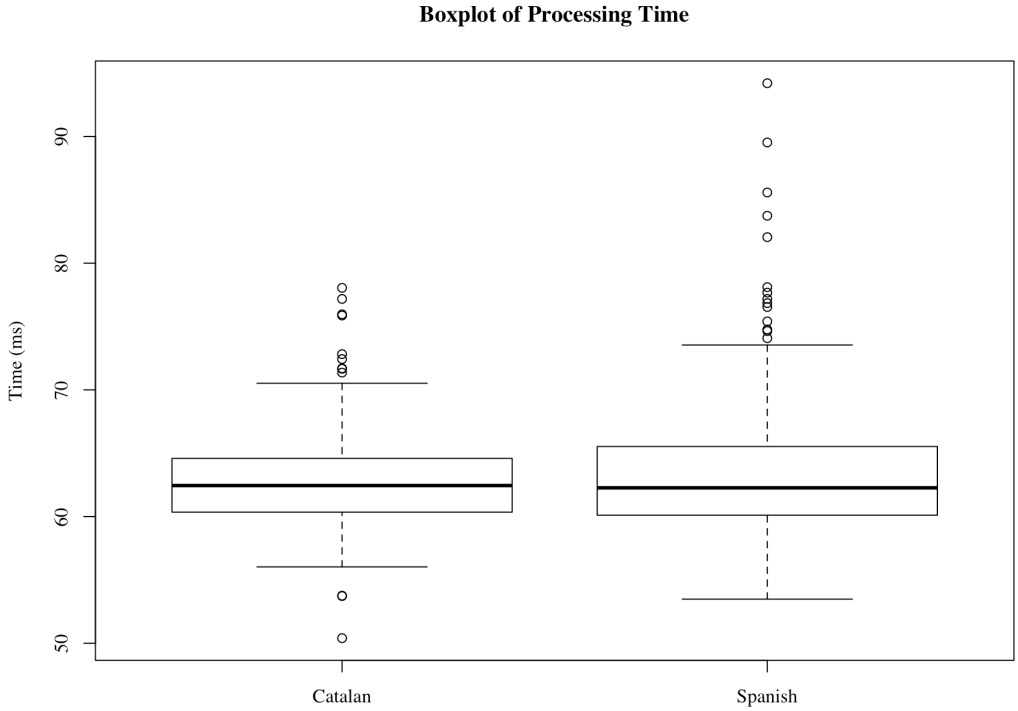

Figure 4.4: Processing Time per Question for each Language

5 | Results Summary

All in all, we finally get a high accuracy for all models, which can vary from 91.9 % to 98.2 % of prediction success, with a global average of 95.25 %. Nevertheless, as explained in last Chapter, these are the accuracies for the questions for which the classification was considered reliable.

	Catalan	**Spanish**
Intent Classfier	91.9	94.2
Topic Classfier	98.2	96.5
Subtopic Classfier	96.0	94.7

Table 5.1: Final Models Performance Applying Threshold (%)

In Table 5.1 we can see the results for all the models, we can observe from the table the following:

- The Topic Classifier is the one that gives better results, with its maximum performance for Catalan, giving a 98.2 % of predictions right.
- Subtopic Classifier made the greatest improve after applying the threshold. Before applying the threshold it was the classifier with lowest results and, after, its performance is close to the Topic Classifier.
- Between languages there is not a big difference. Catalan seems to have a better perfomance for Topic and Subtopic classification, while Spanish performs better for the Intent classification.

We can say that the final results are more than satisfying, giving a high performance for a wide range of questions.

Discriminant Variables

Additionally, we could do an analysis of the words and variables that most affected for the classification. In Figure 5.1 we can see those words for the Spanish Intent Classifiers. The correspondent figures from the other models can be found in the Appendix D, from Figure D.11 to D.15.

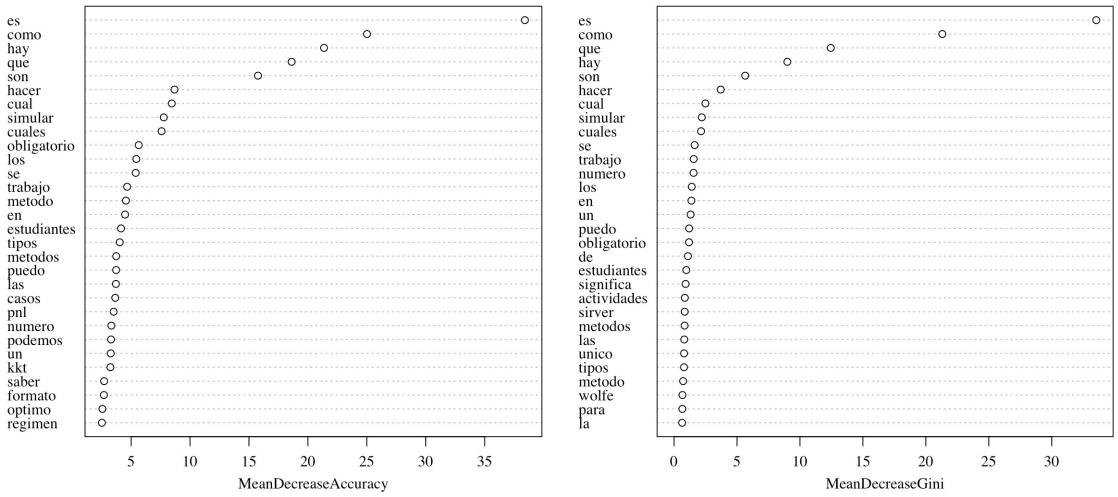

Figure 5.1: Most Discriminant words for Spanish Intent Classifier

For the different kind of classifiers we observed that the following variables were the most discriminant:

- **Intent**. Interrogative pronouns have a big influence on the Intent classification. The variables that represented words such as the translated forms 'how', 'what' or 'which' were among the top discriminant variables. Also the verbs 'to be', 'to do' and 'to simulate' have a big impact on the classification.
- **Topic**. For the Topic Classifier, the most discriminant variables where the frequency of *unique* words from each Topic, which seems logic. Also words such as 'simulation' or 'algorism' are decisive.
- **Subtopic**. Finally, for the Subtopic Classifier, the most important variable is the Topic variable, which indicates the topic of the question. Also the *unique* variables are important again. Concerning words, the most discriminant concept is the translated form of 'system', which is present in several topics.

Between languages there is not a big difference, so the most discriminant words were usually the translated version of the same concept. This result is understandable taking into account the grammatical and lexicon similarities between the two languages.

Part II

Chatbot Implementation

6 | Design Process

Text processing is the core of a Chatbot. But, when implementing it, there are many things that must be taken into account. As a tool addressed to the users, it is primordial to define the kind of public that will use the tool in order to make the interaction useful, friendly and simple. Therefore, during this chapter we will approach the design process of the Chatbot.

6.1 Tool Specifications

First, we must clarify the requirements of the tool, so to propose a solution according to it, capable to meet the requirements at the lowest cost possible. There are two main things that have been considered when thinking about costs, explained next.

Time

We are planning to integrate a real-time tool. Therefore, the time to process each message must be low enough so as not to give the user a sense of waiting. There are three main time limits to keep in mind, which are determined by human perceptual abilities [44–47]:

- **0.1 second** is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result.
- **1 second** is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay.
- **10 seconds** is about the limit for keeping the user's attention focused on the dialogue.

The target of this work is to keep the response time under 1 second, so that the user feels that the flow of messages is uninterrupted. Beyond this limit, we should provide special feedback to the user to maintain the conversation active.

Money

The second important cost to take into account is the monetary, so we will have to provide a functional solution with low expenses. The main expense will be the infrastructural one, as to make the Chatbot operative, it must be continuously active, which requires a consuming server. This will be faced when choosing a server to maintain the Chatbot, in Chapter 9.

6.1.1 Capacity and Target Users

As we introduced in the Introduction's Section 1.1, the Chatbot we are developing is though to serve as a tool for answering questions to students of a specific subject, which has around 300 students per semester. We cannot know the expected amount of users that will be using the tool at the same time. Intuitively, it would be rare to have a concurrency ratio higher than 10 %, i.e. more than 30 students asking to the Bot at the same time. But we will consider the possibility that all students are connected at the same time. Thus, the capacity of the Chatbot should cover the 300 students. Concerning the target users, we can distinguish between two main blocks:

Students

The main target to handle. They will be engineering students in the last years of their degree, aged between 20 and 30 mostly. As said above, it is a group of about 300 people, which its main interaction with the tool will be asking for doubt clarifications from the subject, although we can expect also other kind of behaviors (e.g. greeting the Bot or exploring the limits of its domain by asking off-topic questions).

Professors

The professors of the subject, being one of them the coordinator. This group will be responsible of the maintenance of the Chatbot, together with the developers of it, both handling issues that may occur with the students and/or following up the use made with the tool. Thus, they can be considered the *administrators* of the Bot.

6.1.2 Programming Language

It is also important to chose a programming language that can provide both flexibility and capability during the development of the Bot. Some of the most common programming languages used for Chatbots are [48]:

1. **Python:** Mostly used for it's simplicity. It is also one of the most widely used programming languages in the field of Artificial Intelligence.
2. **Clojure:** A functional programming language that runs on Java Virtual Machine, which enables you to write your app's code as a series of functions which are easy to understand and easy to test.
3. **PHP:** Open source, very easy to use and much faster than other scripting languages.
4. **Java:** It can provide with high-level features for AI projects.
5. **Ruby:** It has a very simple syntax which allows beginners to create a Chatbot easily. However, it might be expensive. It is initially available for free, but at some point it is needed to buy a license.

First, taking into consideration that there are very good open source options, we found appropriate to choose a free programming language. Thus, languages such as Ruby would be discarded.

Second, concerning velocity, it is not primordial to use a very fast language, as our target group is not very big (around 300 users at most). So languages like PHP, C#, or Java are interesting options but not extremely necessary in this case.

After these clarifications, we were looking for a simple, flexible and easy to deal with ML programming language. Python and R seemed to meet these requirements, as they are both similar languages in terms of simplicity and flexibility, and both are widely used. At the end **we decided to use R** because of the following reasons:

- It has a very **good performance** for both ML and text analysis [50].
- The models and text processing (Part I) were already done in R, so it was **easier to fit**.
- Opportunity to **open new doors for future works**. We had already developed Chatbots with Python, but not with R, like many other developers. This is probably due to the fact that originally R was not often used at a production level, unlike Python. However, although R was used primarily in academics and research, it has rapidly expanded into the enterprise market [49]. Therefore, this was a chance to use a language that is widely used in analytics but uncommon for Chatbot solutions in order to evaluate its performance for these kind of solutions.

6.1.3 Interface Languages Available

Additionally, it was important to set the available languages for the tool, as this would impact on the whole implementation process. Originally there were three candidate languages:

- **Spanish**, as it is a common language in the region and should be known by all students that study this subject.
- **Catalan**, also a common language in the region. Moreover, the documentation of the subject is mostly written in this language.
- **English**, used nationally and internationally.

At the end it was decided to develop the tool in both **Catalan** and **Spanish**, as they are the most common language in the region. Moreover, the text analysis is very similar, as they share some grammar and lexicon structure, being both languages derived from Latin.

However, the programming itself of the solution has been done mostly in English and procuring to follow a scalable format, though to make it easy to introduce new languages to the tool, which is proposed for future works.

6.2 Users Management

The tool has been designed mainly for the students of a particular subject. However, as the front-end app (see Section 7) is public, anyone will be able to speak to the Bot. Therefore, we found convenient to design a registration system for the target users. In this Section it is explained how the different kind of users are managed.

6.2.1 User Registration

The registration system has been designed so to meet these requirements:

- Only **students from the subject** are able to access the Bot.
- Students are not obliged to give **any kind of private information**.
- The registration process is **fast and easy**.

Thus, no credentials related to the university can be used for registration, for privacy issues. The final solution proposed is to give a unique *Access Key* (AK) to each student through the virtual campus of the subject.

Thus, the registration process is:

1. Each semester, when the subject starts, ***n* AK are randomly generated**, where n is the number of students from the subject in that particular semester, procuring not to repeated any AK from previous years.
2. The **AK list is stored in a Database (DB)**. This DB will contain the users information (see Section 8.1.2 for the DB structure).
3. When an unregistered user speaks to the Bot, it will be asked to provide an AK. If the user enters a valid AK, **the unique user *id* is linked to that AK**, and the user information is added to the DB (the *id* is provided by the front-end app, as described in Section 7.3). At this time, the user gets a greeting message with some information about the tool functionalities and with the option to change language (see Section 8.3.1 for the language selection functionality).

With this system, the user has to register only once, and s/he does not need to provide any private information. We can see that each student can speak to the Bot only from one account (a single *id*), this seems appropriate in order to control the amount of registered users.

6.2.2 Unregistered Users Handling

As described before, when a user first speaks to the Bot, it is asked for an AK. If a not valid AK is provided, s/he is told to try again. If an unregistered user enters a valid AK that has already been used, the Bot retrieves an alert message saying that AK was previously used and, thus, the user needs to use another AK or talk to an administrator if he or she thinks there was a mistake.

However, in order not to over saturate the system, it was decided to block a user if he or she tries an AK too many times. We concluded that 10 opportunities were enough to enter a valid AK. Thus, an additional DB has been generated for unregistered users. Blocked users can be unblocked by an administrator, so when a user is blocked, an alert message is sent with this information.

6.2.3 Admin Users Assignation

Finally, it was decided that administrator users were needed in order to control the use of the tool. These users will probably be professors of the subject and/or developers. Admins will be able to use functionalities that current users cannot, as described in Section 8.3.2.

7 | Front-End Application

An important matter when implementing the Chatbot is the front-end application where it is going to be integrated. For our case, we decided to use an existent messaging application, so to ease the use of the Chatbot by the students. The following aspects were taken into account when selecting the app:

- **Device Support**. Regarding to the devices that support the app. Knowing our target group, we looked for apps which are available in mobile and/or computer.
- **Channel Usability**. Commonly used channels are prioritized, so to make it easier and more attractive for the users to use the tool.
- **Development Flexibility**. Finally, it is important that the app allows to develop a Chatbot in the most flexible way possible.

Thereby, during this chapter we will first explain the process we followed to select an appropriate app for our Bot and, later, how we created and connected it to our system.

7.1 App Channel Selection

After doing some research about possible channels for Chatbots [51–60], a list of candidates was done:

1. **Facebook Messenger**. Facebook has been the world most popular social network platform for a long time, this makes it one of the channels with larger amount of active users from different age range. Facebook Bots are usually oriented to group chats.
2. **Telegram**. Telegram is a multi-platform instant messaging client created by the Russian entrepreneur Pavel Durov. It currently has 100 million monthly active users distributed around the world, weekly growing 1 million. Telegram is popular for its focus on the users privacy, thing that most of the other instant messaging clients put aside.
3. **Slack**. Slack is an instant messaging client focused on work teams communication. It is one of the most popular tools on its area due to its easy to communicate platform.

4. **WhatsApp**. WhatsApp is one of the largest messaging platforms around the world, with the giant amount of a billion daily active users and it is the dominant app in Spain. However, there is no official way to create WhatsApp Bots nor any API has been provided by WhatsApp to do so, although they are developing a tool to connect users to enterprises.

5. **Skype**. On the same path as Facebook, Skype Bots are generally utilized in group chats for functional purposes.

First, in order to have a better control over the Bot, and knowing the target group, it was decided that the Bot would only be accessible through private conversation, i.e. not in group chats. That makes group-oriented channels not extremely necessary.

Second, WhatsApp was discarded due to the inconvenient that there is no official way to develop Chatbots. Although being the most used messaging app, it is very restricted when it comes to Chatbots, which would have affected negatively during the whole process.

Thereby, a first filtering was done, keeping the most widely used apps that are available in both mobile and computer channels. This leaved us with the platforms Facebook Messenge and Telegram.

At the end, **we opted for using Telegram** as the front-end up for the following reasons:

- Telegram Messenger is **accessible from multiple devices** (mobile, computer, tablet) **and platforms** (Android, iPhone, iPad, Microsoft Windows, Web-version, macOS, PC, Mac, Linux), which makes it very reachable. You can look all available platform apps at Telegram's Home Page[1].

- Although it has less active users than Facebook Messenger, it still has got 62 million daily active users and it is growing at a rate of more than 50% annually, being **one of the most common messaging apps** both worldwide and in Spain.

- Finally, and the most decisive factor, is **its flexibility**. The Telegram Bot API (see next Section) has got multiple functionalities that enriches the user experience, provides a huge range of possibilities when developing a Chatbot and allows us to implement all the text processing previously done in Part I.

Having said that, we are going to describe now how we created the Bot with Telegram and the functionalities that have been useful for our use case.

[1] `https://telegram.org`

7.2 Chatbot Creation

Telegram has got a specific Bot for creating other Bots, called *BotFather*. Thus, a developer can create a Telegram Bot by chatting with the *BotFater* (to learn more about how to create and set up a Bot, please consult Telegram's Introduction to Bots[2] and Bot FAQ[3]). To do so, two parameters are asked:

- **Bot's *Name***. That is, the name that will be displayed in a conversation with the Bot. In our case, we named it ***Liam***. The name was selected as an allusion to the mathematician William Karush and his successors Harold William Kuhn and Albert William Tucker, who published the Karush–Kuhn–Tucker (KKT) conditions [61], present in the subject in scope. The three of them share the name *William*, so we decided to use its diminutive *Liam* for our Bot in their honor.
- **Bot's *Username***. This is the *username* for which the Bot can be searched in Telegram. It is unique, i.e. no other Bots or users in Telegram will have this *username* and, for Bots, it must end with 'bot'. We decided to call it ***optisim_ bot***, as OptiSim is the acronym of the subject in scope and a common name used by the students to refer to it. Thus, we decided to give this *username* so that students could remember it easily.

Once this parameters are passed, Telegram retrieves a success message containing a *token*. Each Bot is given a unique authentication *token* when it is created. The *token* looks something like `123456:ABC-DEF1234ghIkl-zyx57W2v1u123ew11`, but we'll refer to it simply as ***TOKEN*** from now on.

This *TOKEN* is needed for the HTTPS requests to Telegram's Bot API (see next section), and it is the identifier from which you can control the Bot. Therefore, it is a variable that must be kept private, that is why it won't be present in this document.

Figure 7.1: Bot Initialization Chat Page

[2] `https://core.telegram.org/bots`
[3] `https://core.telegram.org/bots/faq`

7.3 The Telegram Bot API and Linking from R

As described at the Telegram Bot API Documentation Page[4], the Telegram Bot API (from now on called only Bot API) is an HTTP-based interface created for developers keen on building Bots for Telegram.

Thus, the way to control a Bot is by using HTTP requests to the Bot API, indicating the Bot's TOKEN. In this section we will explain how this requests work and how we linked the Bot from R.

7.3.1 Bot API Requests

There are two mutually exclusive ways of receiving updates for a Bot:

- **Polling**. The developer can get updates by sending an HTTPS GET request to the Bot API, specifying the Bot in scope, through the `getUpdates` method. Also Long Polling is supported (this concept is explained below).
- **Webhooks**. Through this method, the developer must specify a URL and receive incoming updates via an outgoing webhook. Whenever there is an update for the Bot, Telegram sends an HTTPS POST request to the specified URL, containing the update.

Updates come as JSON-serialized objects, and they contain the information about the messages sent to the Bot. They are stored on Telegram's server until the Bot receives them either way, but they will not be kept longer than 24 hours.

For our case, we decided to use Polling instead of Webhooks, as it is easier to set up and works well for what we need. To be precise, though, we actually used Long Polling, a variation of the traditional Polling technique.

Long Polling Usage

When you intent to build an active Bot it is not recommended to massively do web requests, as it is not great for Telegram's servers (they explicitly ask people not to do this outside of testing scenarios) and not great for our resources either. Long Polling takes advantage of the fact that most of the time, the Bot is receiving "empty" responses. As our Bot is probably not going to be receiving messages every half second, most of the times that the updates are asked, there aren't any [62].

[4] `https://core.telegram.org/bots/api`

With Long Polling (LP) [63], instead of Telegram telling that there aren't updates, it simply keeps the connection open until there are updates, and then sends these down the open pipe. As it is impractical to keep a connection open forever, the number of seconds that the system will we waiting for can be specified. This is done by passing another optional argument to the `getUpdates` call named `timeout`.

Therefore, the Chatbot will be active through a main loop that will contain the following steps:

1. **Calling the `getUpdates` Method**. First, an HTTPS request is done to Telegram using LP through the `getUpdates` method. This will open a connection with Telegram until the URL status changes or the time specified through the `timeout` argument passes. In our case we set this argument to 5 minutes, after that time, a new call is done.

2. **Receive Message**. When a user writes to the Chatbot, the status of the URL changes and thus the actual GET request is done, retrieving the message update information, which contains, among others, the unique Telegram's user *id* that sent the message.

3. **Message Process**. This message is processed as described all along Parts I and II of this document, and an answer is generated.

4. **Send Response**. The generated answer is sent to the user through another HTTPS request sent to Telegram, specifying the user *id* to whom the response is addressed.

5. **Back to Step 1**. The loop starts again.

7.3.2 An R Wrapper Around the Telegram Bot API

In order to manage the Bot from R, we have used the R package `telegram`, created and developed by Luca Braglia [64]. It allows you to create a `TGBot` object, which derivates from `R6` classes, and uses other packages such as `curl`, `jsonlite` or `httr` in order to ease the access to Telegram's messaging facilities (e.g. getting updates or sending messages, images, files from R to Telegram).

Moreover, the author has also written a quick guide on how to use the package in his GitHub Repository[5], where the developer version of the package is stored. You can also find further information about the package in its CRAN Reference Manual Documentation[6].

[5] `http://github.com/lbraglia/telegram`
[6] `https://cran.r-project.org/web/packages/telegram/telegram.pdf`

7.4 Updating the R Package 'telegram'

When we started developing the back-end, we noticed an important lack in the `telegram` R package. That is, that the package's `getUpdates` function was not programmed to accept the `timeout` argument and, thus, this parameter could not be passed in the LP request to Telegram. When this happens, the API sets its default `timeout` value, which is 0, i.e. Short Polling. This left us with the Polling problematic explained previously.

Adding this to the fact that some user-enriching functionalities that we pretended to integrate were also not implemented in the package yet, we decided to update it with the following:

- **Adding `timeout` Argument**. We added the `timeout` input variable to the `getUpdates` function, so to pass it trough the HTTPS request when using this function and, thus, using the LP method.
- **Keyboard Displaying**. Functions that allow in-chat keyboard displaying have been added. This includes adding input variable `reply_markup` to the `sendMessage` function, as the Bot API sets the keyboards through that argument. Also the keyboard objects supported by this argument have been added:
  - `ReplyKeyboardMarkup`. This object represents a custom keyboard with reply options. Used in our case for the operator quick responses, for instance (see Section 8.2.5).
    - `KeyboardButton`. This object represents one button of the reply keyboard, which has also been created.
  - `InlineKeyboardMarkup`. This object represents an inline keyboard that appears right next to the message it belongs to. We use this kind of keyboards for the language selection and the Multi-choice AF (see Sections 8.3.1 and 8.2.4 from Chapter 8, respectively, for more information).
    - `InlineKeyboardButton`. This object represents one button of an inline keyboard, which has also been created.
  - `ReplyKeyboardRemove`. Upon receiving a message with this object, Telegram clients will remove the current custom keyboard and display the default letter-keyboard.
  - `ForceReply`. Upon receiving a message with this object, Telegram clients will display a reply interface to the user (act as if the user has selected the Bot's message and tapped 'Reply'). This was not needed, but as it was the only object remaining in this area, we decided to include also in case someone else wanted to use it.
- **Adding `answerCallbackQuery` Function**. After the user presses a callback button from an inline keyboard, Telegram clients will display a progress bar until the `answerCallbackQuery`

method is called. It is, therefore, necessary to react by calling `answerCallbackQuery` even if no notification to the user is needed (e.g., without specifying any of the optional parameters). Thus, we had to create this function in order to send answers to callback queries sent from inline keyboards.

- **Adding `sendChatAction` Function**. This method sends to the user a status follow-up when something is happening on the Bot's side. We use it when sending documents to the user. As it can take some seconds to send a document, before doing so we send this kind of action to the user so to notify that a document is being sent. Thus, the user doesn't get a sense of chat inactivity.

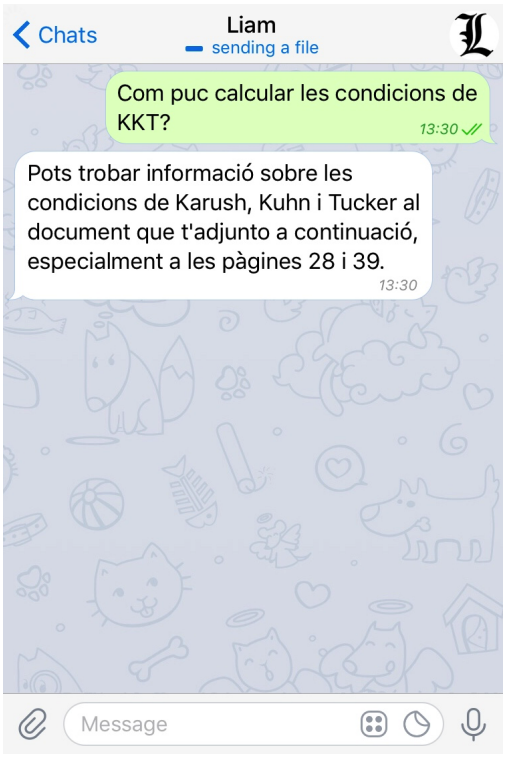

Figure 7.2: Chat Action *sending a file*

All the features have been properly documented following the R CRAN standards, and a pull request has been sent to the package author so to implement these functionalities into the stable version of the package.

In the meantime, as these functionalities were needed for the development of the back-end, we decided to upload the updated version of the package in our own GitHub Respository[7], which is forked to the author's package one (see Section 7.3.2).

From R we can download the package from that source through the `devtools` package function:

```
install_github('ebeneditos/telegram')
```

This way, when importing the library `telegram` we will load our updated version of it.

[7] `https://github.com/ebeneditos/telegram`

8 | Back-End Development

Once explained the design process and having an appropriate front-end application to support our Bot, we will proceed to explain how it works in the back-end. To do so, we will:

- First, detail how all the data needed to run the Bot was structured and explain the two environment used for developing.
- Second, we will take a look at the different Answer Formats that our Bot can retrieve.
- Third, a listing of other functionalities implemented is done.
- Finally, the error handling is treated, explaining the log generation and the error catching.

8.1 Data Structure and Set-up of the Environments

In order to keep the code clean and organized, we have structured the documents so to ease the development of the tool and facilitate possible external use of it. In the Appendix C: Scaling the Solution you can find the way to scalate this solution to other use cases.

8.1.1 Data Structure

First, are going to detail how the data has been structured. The following files and directories can be found in the root directory of the Bot:

8.1.1.1 `Liam.R` script

This is the main script, where the other scripts are loaded, the Bot object is created and the main loop function is called.

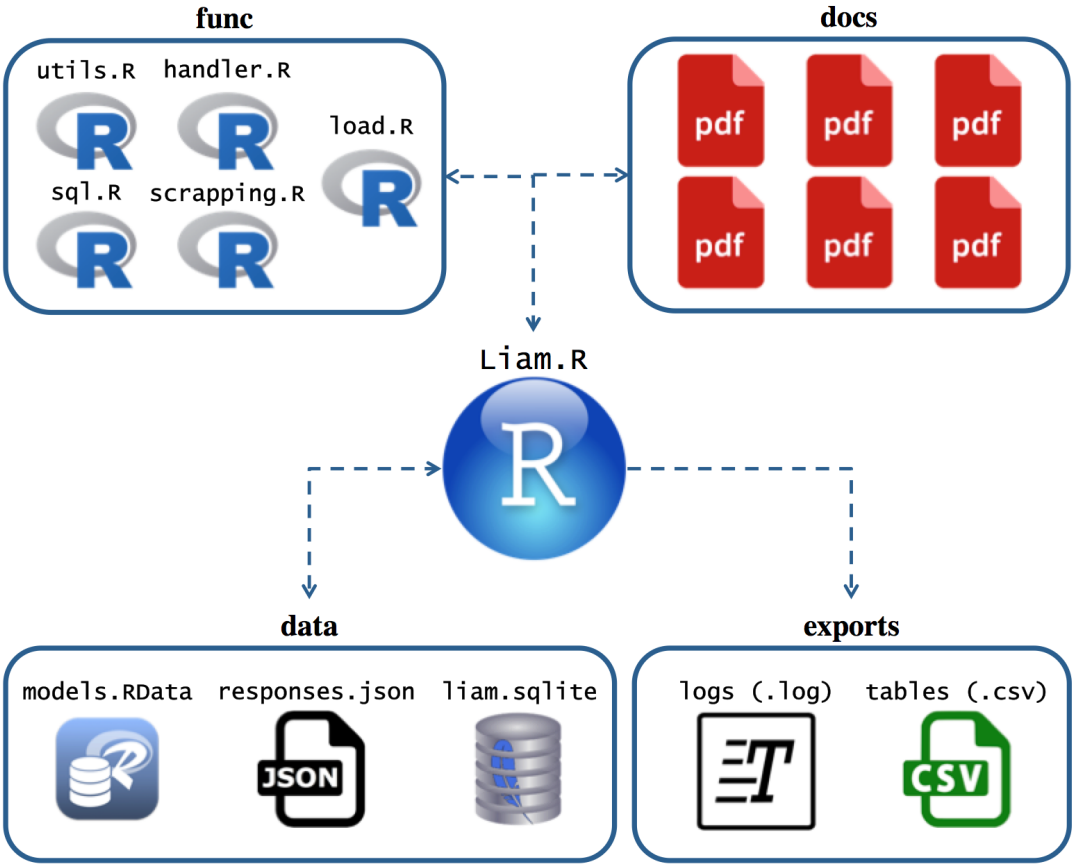

Figure 8.1: Diagram of the Data Structure

8.1.1.2 `func` directory

Here we store the auxiliary scripts that contain all necessary functions to make the Bot work. These scripts are:

- **`load.R` scrip**. This script loads all necessary packages and also the functions from the following scripts. It is called from the main script.
- **`utils.R` scrip**. It contains the core functions of the program, including:
  - **Main Loop function**. Function that starts the loop described in Section 7.3.1.
  - **Text Processing functions**. Needed for the text processing.
  - **Get and Set functions**. Which allows to get information and set parameters.
  - **Update functions**. For updating the DB.
  - **Log functions**. For the generation of the logs.

- `handler.R` **scrip**. We structured the message and answer handling with easy-to-scalate functions present in this script. The Answer Formats handling, explained later in Section 8.2, is mainly done trough the functions of this file. New functionalities can be easily added following the structure of this script.
- `sql.R` **scrip**. With functions that allows the connection to SQLite DB.
- `scrapping.R` **scrip**. Contains functions used for the document scrapping, used for the information answer retrieving (explained previously in Section 3.6).

8.1.1.3 `data` directory

We can find the necessary data for the running of the Bot, including:

- `liam.sqlite`. It is the SQLite DB which contains the users and messages history tables. You can find additional information about the SQLite DB connections in Section 8.1.2.
- `models.RData`. The file with the classification models trained. As they are an external file which is loaded only when the Bot is started, models can be re-trained even if the Bot is running and, once we have the new models in this folder, we can reload them with the Bot command `update`, which is explained in Section 8.3.2.
- `responses.json`. Contains the default answers in different languages, its structure looks like:

```
{
"ResponseKey 1":  {
     "Language 1":  "Response 1 for Language 1",
     "...":  "...",
     "Language m":  "Response 1 for Language m"},
"...":  {...},
"ResponseKey n":  {
     "Language 1":  "Response n for Language 1",
     "...":  "...",
     "Language m":  "Response n for Language m"}
}
```

  Therefore, we have created a `ResponseKey` for each of the n default responses of our system. In our case we provided responses for $m = 2$ language, Catalan and Spanish. This object has been loaded as a `list` object called `ResponseDispenser`, so it is easy to get

the answer from it. For example, if the system detects that the user sent him a greetings message in Spanish, the Bot will respond as well with a greeting message, which will have the parameters `ResponseKey = "Greeting"` and `Language = "Spanish"`. Thus, the text of the response can be get as `ResponseDispenser[[ResponseKey]][[Language]]`.

With this structure it is easy to get the appropriate answer and also to add new response languages to the system.

8.1.1.4 `docs` directory

Containing the `pdf` files of the subject's documentation, needed for the information retrieval answers.

8.1.1.5 `exports` directory (automatically generated)

This directory is automatically generated when the Bot starts. It is an only write directory, which will contain the following:

- **Logs**. The logs files generated to keep control of the session, described in Section 8.4.
- **DB Tables**. The admins of the Bot can export the tables stored in the SQLite DB through the command `export` (explained in Section 8.3.2). When this is done, the system exports a `csv` file for each of the tables to this directory.

8.1.1.6 `update` directory (only developers)

There is an extra directory that will be only present in the developers environment (i.e. not the production one). In this directory there are two subdirectories:

- `sqlite` **subdirectory**. With scripts and files useful for changing, deleting or adding tables to the SQLite DB.
- `models` **subdirectory**. With training functions for the model. An automatic function has been developed that trains and tunes the parameters of the three RF classifier models needed for each of the languages of the tool. We only need to update the file containing the training dataset and run the script to do so.

Thus, in this directory we can create new `liam.sqlite` and `models.RData` files which can replace the ones stored in the `data` directory, updating this way the Bot.

8.1.2 Connecting to SQLite Database

There are several tables that must be updated frequently while the Bot is running. To do so, we decided to manage our DB through the R package `RSQlite`. This package embeds the SQLite database engine in R, providing a DBI-compliant interface. SQLite is a public-domain, single-user, very light-weight database engine that implements a decent subset of the SQL 92 standard, including the core table creation, updating, insertion, and selection operations, plus transaction management [65].

With this package, we can create a relational database `.sqlite` file and open a connection to it from R, allowing us to make queries and modifications to the tables stored in that DB. In our case, we created the following tables:

- **users**. Containing information about the registered users, with information about:
  - **Access Key**. Needed for the registration. Originally this field is already filled, with all AK generated for registration purposes. When a user enters a valid AK, his or her user *id* is linked to it.
  - **Language**. User selected language. If not specified, it is Catalan by default.
  - **Number of Messages**. Number of messages that a user has sent.
  - **Telegram User Information**. Information about the user is stored, such as its *id*, among others.
- **blacklist**. Containing information about users that talked to the Bot not being registered. In this case, when the number of messages sent by a user is 10, that user is blocked.
- **messages**. With the history of messages sent to the Bot, containing information about:
  - **Time**. Time variables informing when the message was sent.
  - **Chat**. Information about the chat and the user that sent the message.
  - **Message**. Variables about the content of the message.
  - **Response**. Informing about the kind of response the Bot sent back.
- **replies**. We did an additional table storing the messages that were forwarded to an operator (the forward-reply structure is explained in Section 8.2.5).

8.1.3 Setting up the Environments

Having a second environment is very useful for developers in order to update the system without stopping the main Bot. Therefore, we have defined the following environments:

- **Production Environment *(Prod)***. The production environment, i.e. the environment where the main Bot is run, with all the necessary files to do so.
- **Testing Environment *(Test)***. It has a copy of *Prod* files plus additional files used for development purposes.

Both environments are stored in a private GitHub repository project, which has been configured with the GitHub Education[1] facilities, linking the account to the university address. This way, the process when updating the Bot is:

1. **Developing on *Test***. The developers of the tool can change the *Test* code and files in order to make the new implementations. *Prod* remains intact and the main Bot is running with the stable version.
2. **Testing New Implementations**. Once the developing has been done, we can test the new implementations. To do so, we decided to create a second Bot, with parameters:
   - ***Name*:** *Ewan*.
   - ***Username*:** *ois_test_bot*.

   Thereby, the main Bot can still run with the stable version.
3. **Stabilizing *Test***. Through testing, we can correct bugs and structure properly the code until the *Test* version of the Bot is stable.
4. **Pushing *Test* updates to *Prod***. Finally, we can push the new implementations to *Prod* and restart the main Bot.

[1] `https://education.github.com`

8.2 Answer Formats

In Section 3.6 from Part I we saw the answers retrieved to the user when an Intent and a particular Entity were identified by the system. However, other Answer Formats (AF) have been defined for different kind of necessities.

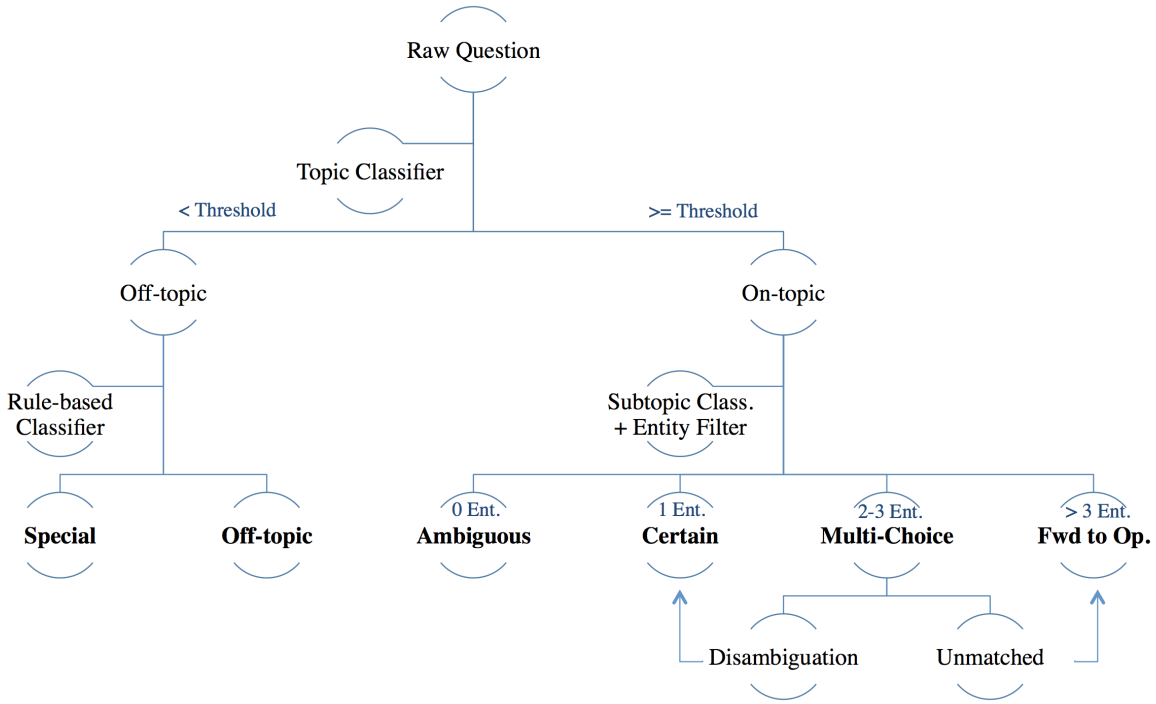

Figure 8.2: Answer Formats Decision Tree

In Figure 8.2 we can see the different possibilities when handling an answer. In Part I we focused in the *Certain* AF. However, there are multiple Answer Format scenarios, such as:

- **Off-topic**. When the question is set as off-topic, a different methodology is followed.
- **Ambigous**. Ambigous question.
- **Certain**. When the system gets one and only one Entity. This is the only AF where the AT approach is needed.
- **Multi-choice**. When the system detects more than one Entity, it interacts with the user in order to delimit the question.
- **Forward to Operator**. Answers that are forwarded to an Operator.

Below we proceed to explain each of them.

8.2.1 Off-topic

When a question is set as off-topic, a rule-based classifier is applied, and the possible answers are:

- **Special**. Special answers are retrieved if the rule-based classifier detects one of the following:
  - **Greetings**. The user is saying some kind of *Hello* message.
  - **Farewell**. When the user is saying goodbye.
  - **Gratification**. When the user is responding a kind of *Thanks* message.
- **Off-topic**. If none of this is detected, the system responds a default answer asking the user to talk about its domain.

8.2.2 Ambiguous

If the question is on-topic, then the Subtopic Classifier and Entity filtering is done (see Section 3.5.2). However, there are cases where, after this process, no Entities are found compatible. In this case, the system responds the user asking him or her to be more specific.

8.2.3 Certain

On the other hand, if the system finds one matching Entity, the system answers the appropriate AT with the desired information. This is the main case for which the Chatbot has been built, and the process for this AF can be found in Section 3.6 from Part I.

8.2.4 Multi-choice

However, although unusual, it can be the case that more than one Entity is matching. In this case, we decided not to provide information of all Entities, as that would mean heavy answers, which are not optimal for a chat.

Instead, a multi-choice AF has been designed. Thus, the system would respond with a list of possible Entites plus one final option to indicate that none of the Entities listed are right. Therefore, the answer of the user to this kind of AF is re-directed as explained below:

- **Disambiguation**. The user picks one of the Entities listed, then the system responds to it with the **Certain** AF.

- **Unmatched**. If the user picks the last option, indicating that none of the Entities listed is correct, then his or her initial question is **forwarded to an operator**.

In order not to give a too long list, we decided that Multi-choice AF is appropriate when the system has detected a maximum of 3 Entities. If more Entities are detected, the question is automatically forwarded to an operator.

8.2.5 Forward to Operator

In some cases, the question is too complex for the system and extra process has to be done. This happens for this two cases:

- User picks last option in a **Multi-choice** AF.
- More than 3 Entities are detected.

Thus, we decided to forward this kind of questions to an Operator. This was discussed and accorded with the coordinator of the subject, as he acceded to carry out this task. Therefore, the flow of this AF would be:

1. System Answer Handling process sets this AF and, thus, the user question is forwarded to the chat that the Bot has with the Operator.
2. The Operator receives the forwarded question and responds to it. Some quick responses have been defined for the Operator in order to respond in one click:
    - **Inappropriate**. The question is not appropriate and thus the system sends an alert to the user.
    - **Off-topic**. The question is not from the system domain and, thus, the user gets an answer with AF **Off-topic**.
    - **Answer Later**. The Operator indicates that the question is appropriate but is not able to respond at the time. The user would get a message saying that the question cannot be processed at the time.
    - **Answer by Hand**. Otherwise, the operator can respond by hand to the answer.
3. Finally, the user gets a chat reply to his question with the subsequent response .

8.3 Additional Functionalities

Furthermore, some additional functionalities for the users have been implemented in order to enrich the experience or ease the management of the tool by the administrators.

8.3.1 Language Selection

In the first place, as the tool is available in two languages, Catalan and Spanish, a functionality has been programmed so that any registered user can change the language at anytime. To do so, s/he just needs to enter the initial command `start`, which is obligatory to use when starting a conversation with the Bot, and the Chatbot will ask if the user wants to switch language.

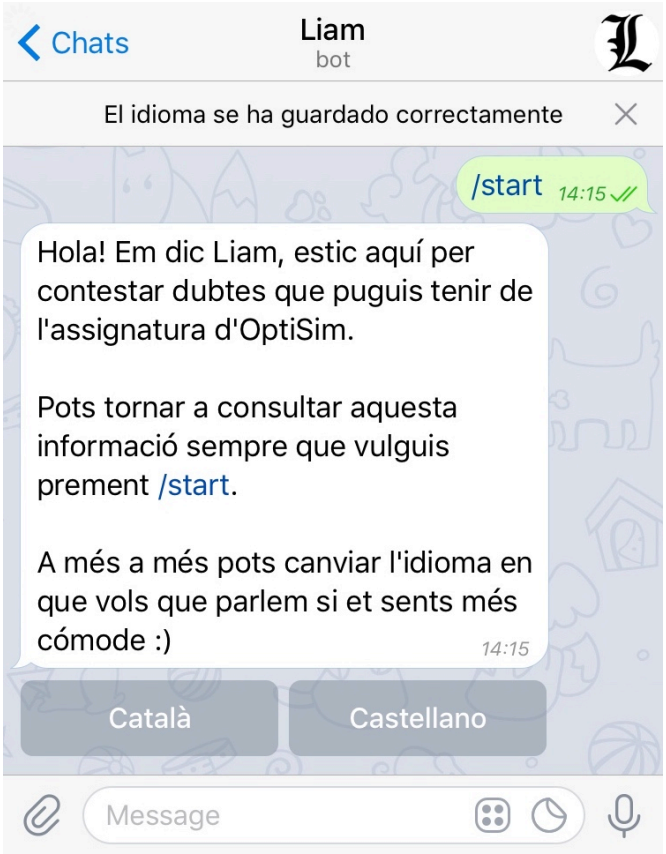

Figure 8.3: Greeting Message and Language Selection

In Figure 8.3 above we can see the typical welcome message in Catalan sent from the Bot and the language selection displaying. When the user picks one of the options, a success banner is retrieved to him or her in the selected language, Spanish in this case.

8.3.2 Admin Commands

Additionally, some functionalities have been developed for admins. They can use them by typing the appropriate commands to the chat:

- `export`. Writes the users and messages DB into `.csv` files.
- `resetuser`. Allows to reset the `user_id` associated to a particular AK. This is useful for the case that a user has registered with a AK that does not belong to him/her.
- `unblock`. Unblocks the user with the `user_id` associated.
- `update`. Reloads the models and responses files.
- `pause`. Pauses the Bot. This closes the connections from the Bot to DB and sends a default message to any user intending to chat with the Bot informing that it is in pause for maintenance purposes.
- `resume`. Resumes the paused Bot. When it is done, questions received during the pause are replied.
- `kill`. Stops definitively the Bot in a safe way.

To call any of these commands, the admin just needs to send a message trough the Bot chat with the command preceded by a *slash* (/). Some of this commands need an input argument to work, which are:

Command	**Argument**	**Description**
`export`	*tables (optional)*	You can specify which tables you want to export. If this argument is not specified, then all tables from DB are exported.
`resetuser`	*key*	The key password of the user you want to reset.
`unblock`	*user_id*	The unique Telegram *id* from the blocked user.

Table 8.1: Admin Required Command Arguments

Arguments are passed leaving whitespaces between them. Thus, if we wanted to unblock a user with `user_id = 12345678`, we could do it by sending to the Bot the message:

/unblock 12345678

If a user which is not an admin tried one of these commands, he or she would get a response saying that command is not available.

8.4 Error Handling

When the Bot is running, some errors may occur (e.g. connection with Telegram failed). In order to keep truck of them we have made some error handling, which allows to catch and prevent them.

8.4.1 Log Generation

In the first place, we have generated two log files, where information about the session and the errors are stored. When the Bot is initialized, a connection is open with such files:

- `liam.log`. This keeps truck of the usage of the Bot. Thus, whenever a user registers, sends a message or changes its preferences, this log is updated. Also admin commands and other changes are registered. Thereby, by looking at this log one can have a general idea of what was done with the Bot while it was running.
- `session.log`. In this log we register all errors that may occur. This is done through the library `sink`, which automatically writes the error outputs to the linked file. This way, through this document we can check what failed during execution.

These files are stored in the `log` directory inside `exports`, which is created during the Bot initialization.

8.4.2 Error Catching and Prevention

When having an active Bot running, it is not convenient that errors stop the execution, as that would break the whole process.

In order to prevent this, we used the R function `try`. This is a wrapper to run an expression that might fail and allow the user's code to handle error-recovery. The function `try` is implemented using `tryCatch`, and it evaluates an expression and traps any errors that occur during the evaluation [66].

Thus, functions that originally do not retrieve a response in our code, which are the majority of them (e.g. writing to SQLite DB, handling a message or sending an answer) were set them to return `TRUE` and thus evaluated through the `try` function. This way, if there was no mistake, the evaluation variable of the function would be `TRUE`, and otherwise it will print the error in the log and continue with the process.

9 | Moving to the Cloud

During the development of this project, the testing of the Bot has been done running it locally. Though, as we pretend to give an unstopped service for the students of the subject, we had to find feasible ways to do it, meeting the following criteria:

- **Capacity**. The host server must be able to fill the capacity of users that are expected to use the tool.
- **Expenses**. The monetary expenses should be as low as possible, so to make the implementation affordable.

However, as the Beta phase of the Bot is expected to initialize after the project ends, and it was not considered inside the scope of it, we did not test the server at the ending of the project. Although, we did the necessary managements to find an appropriate server for this purpose.

Server Selection

Together with the coordinator of the subject, we asked to the university if they could let us use its servers. After some search, we ended up contacting with the UPC's *Institut d'Organització i Control de Sistemes Industrials* (Institute of Industrial and Control Engineering), which is abbreviated IOC. As a university institute, its functions are research and technology transfer, as well as the teaching of postgraduate courses. The institute has, among its other laboratories, a computer network equipped with servers, workplaces, PC's and software [67]. You can find further information at the IOC Home Page[1].

The IOC kindly let us use their servers for free. These are DELL PowerEdge R530 with 64 GB RAM and 2 Intel Xenon 2630 v4 CPUs servers, which seem to fit into the capacity criteria. Therefore, we considered this was an excellent option for our purpose and we expect to start the server testing during the following weeks after this project is finished, when the Beta Phase starts.

[1] `https://ioc.upc.edu/en`

End-to-End Process Structure

Nonetheless, we did design the structure of this server so to make the code updating easy. We decided to create a GitHub account for production purposes, which has read rights to the *Prod* environment from our repository, which was introduced in Section 8.1.3. Therefore, whenever there is an update posted in the GitHub repository, we can download it from the server by pulling from the repository.

The setting up of the Chatbot can be summarized with the following steps:

1. **Set-up the Server Account**.
2. **Download R and RStudio**.
3. **Create a Git Version Control RStudio Project**.
4. **Set Global Variables**.
5. **Run the Bot**.

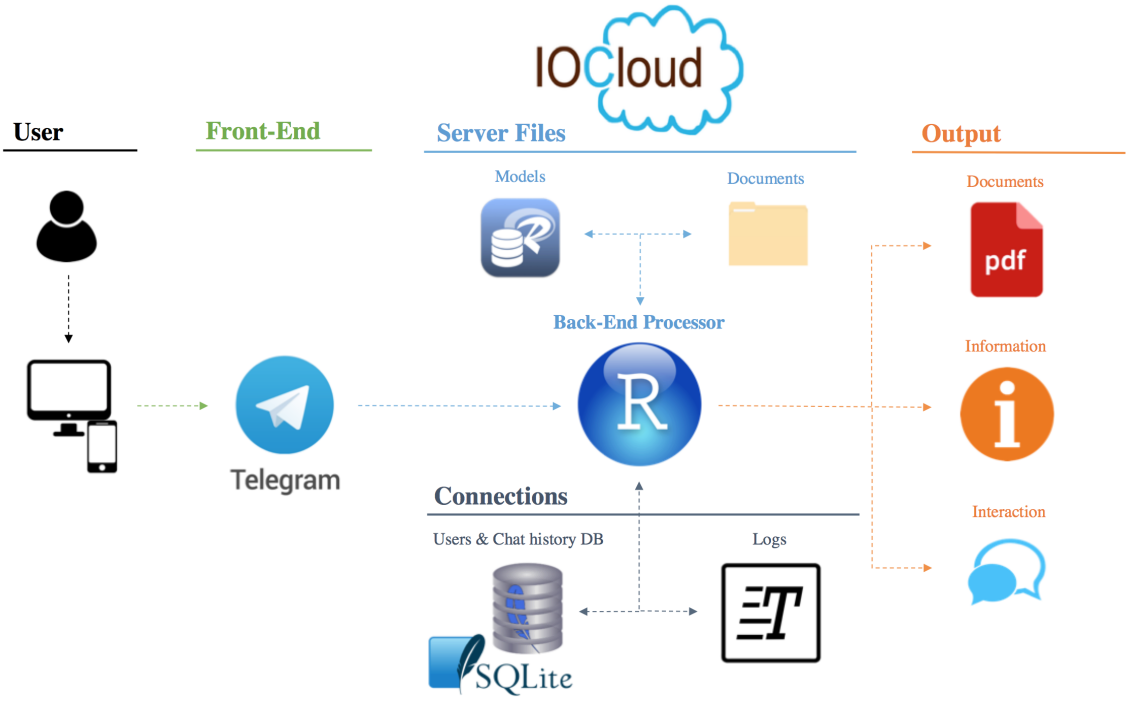

Figure 9.1: Chatbot Flow Structure

In Figure 9.1 we can see the the overall flow of the end-to-end Chatbot structure, finding also the summary of files that can be found in the IOC server.

Conclusive Part

10 | Conclusions and Future Work

By doing this project, many things can be said about what was learned and there are some tips that could help people willing to so similar projects. In this final chapter, the conclusions are exposed, as well as the contributions made in this project and the future work proposed.

10.1 Conclusions

The overall outcome of the project has exceeded by far the initial expectations. We were able to accomplish the initial goals and do more than what originally was in scope. First, we will make a recap on the different aspects of the project done:

- **Text Understanding**. We have been able to build a system that is able to understand the questions in two languages, generating up to 6 different ML models that have an average accuracy higher than 95 %, which performs well for our solution. An unexpected additional effort had to be done by generating from scratch the dataset, with the collaboration of the coordinator of the subject.
- **Answer Generation**. Initially we intended to do a pre-built kind of answer, i.e. a predefined answer for each potential question. We finally generated an hybrid Answer Type structure that is also able to retrieve parts of documents or text from them.
- **Front-end Application**. We have been able to implement this system into an actual text messaging application, Telegram, which is widely used, thus giving the opportunity to reach our system a huge range of users.
- **Back-end Development**. We have implemented a wide range of functionalities for our Chatbot, which enriches the user eperience and provides a useful service. Also a big programming architecture has been developed, which permits the expansion of the actual solution and is easily scalable to others.
- **Implementing the Solution**. Moreover, we have routed a Beta Phase in order to test the solution with actual students, which will be explained during the Future Work Section.

- **R performance**. Once the solution is fully operative, we can conclude that R is a good option for Chatbot solutions, given all what has been seen during the project, both for its flexibility and the performance provided for text processing tasks.

Additionally, we also had to design several interconnected structures and functions, such as:

- User registration system.
- Generation of interactive answers for the users, including an operator forward-reply functionality.
- Language selection.
- Professor functions so to control the tool.
- Capacity to re-train without having to stop the Bot.
- Ability to update the code easily with the use of two environments.
- DB and error connections and management.

Notwithstanding, many things took a higher effort than the one initially expected that have introduced an over-cost to the whole project, including the updating of the R package 'telegram', among others.

Furthermore, a considerable autonomous learning has been done all along the project, which has provided lots of knowledge about many different fields, for which we are satisfied.

Also, during the project we found convenient to generate extra documentation of the project for further development. That is why we generated 3 documents specifying how to access the Bot, how to load its sources locally and how to scale the solution to other problems, which are included in Appendices A - C.

Looked with perspective, it feels like we could have dedicated an entire Master Thesis to the text understanding part, another one for the answer generation and another one for the implementation. So if someone pretends to do a similar work, we have to say that it requires to take into account multiple aspects and take lots of decisions. Of course, we have not been able to enter to all of these aspects at a full level of detail, but we did a great effort to build an overall solution with a high performance, considering the magnitude of this thesis.

All in all, we are very pleased with the work done, as we have been able to build an end-to-end solution for the area of education to serve both students and professors, introducing an innovative way to teach.

10.2 Contributions

Whit this project, four main contributions have been done for further research.

- **Innovative solution provided**. We have provided an end-to-end solution to the field of education, which has little precedent in the field.
- **Open new doors**. With his solution and this project, we expect to open new doors for the interaction within the educational world.
- **R Chatbot precedent**. Furthermore, we have created a precedent with the development of a Chatbot using R, as it is a widely used programming language in analytics, but not yet often used for these kind of solutions.
- **'telegram' package update**. Finally, with the updates made in the R package 'telegram', we expect to ease the development of such Chatbots and to improve the user experience of Telegram Chatbots developed with R.

10.3 Future Work

Finally, we want to mention the two lines of future work that may be done. In the one hand, there is the future work that will be done in order to start the Beta Phase of the Chatbot with a real use case. This includes moving the Bot to a server and testing its capacity, as well as doing a six month follow up of the progress of the phase. On the other hand, as we said there are many things that could be researched with more detail. The most important we have found are:

- **Text Processing**. The use of deep NN could be proposed for the text classification, as well as an automatized KW detection. A larger datast would be recommendable for it.
- **Answer Generation**. Generative answers can be a distinctive factor of a Chatbot. Further research on how to generate them would enrich the whole system.
- **Chatbot Implementation**. For our target group the Chatbot implementation was appropriate. It would be useful though research on implementations for bigger scales, using techniques such as parallelized code. Also other complementary functionalities such as speech-to-text could be proposed.

We hope that with this project others feel motivated to continue with the work. For any doubts you may have, you can contact the author of the project, Ernest Benedito, via e-mail[1], or you can check his GitHub Respository[2], where code used during this project will be shared.

[1] `ebeneditos@gmail.com`
[2] `https://github.com/ebeneditos/TGBot`

Appendices

A | Installation Guide

In this Appendix it is explained the process to follow in order to talk with *Liam*, the Chatbot developed during this project.

1. **Download Telegram**. You can do it from *AppStore*, *Google Play*, or from Telegram's Home Page[1].

2. **Search the Bot**. Inside Telegram, go to *'Search for messages or users'* and type in *optisim_bot*. The name of the bot should be *Liam*.

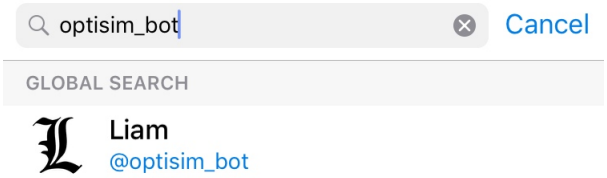

Figure A.1: Chatbot Username Search

3. **Enter a Valid *Access Key***. Inside the conversation with *Liam*, press *Start*. The Bot will ask you for an *Access Key*, you can get it in the virtual campus of the subject (if you are a student of it, of course).

4. **Chat!** It is time to start tolking with *Liam*. Saying 'Hello' is always polite.

For any doubts you may have, you can contact the author of the project, Ernest Benedito, via e-mail[2], or you can check his GitHub Respository[3].

[1] `https://telegram.org`
[2] `ebeneditos@gmail.com`
[3] `https://github.com/ebeneditos/TGBot`

B | Developer Manual

In this Appendix it is explained the process to follow in order to load the Bot to your local environment, you should follow the next steps:

1. **Download R and RStudio**. You can do it from the CRAN[1] and RStudio[2] web pages.
2. **Create a Git Version Control RStudio Project**. In order to download the sources of *Liam* form the GitHub repository they are located, it is recommended to create a Git VC project in RStudio, you can do it by:

   (a) Inside RStudio, go to *Project* > *New Project...* > *Version Control* > *Git*.

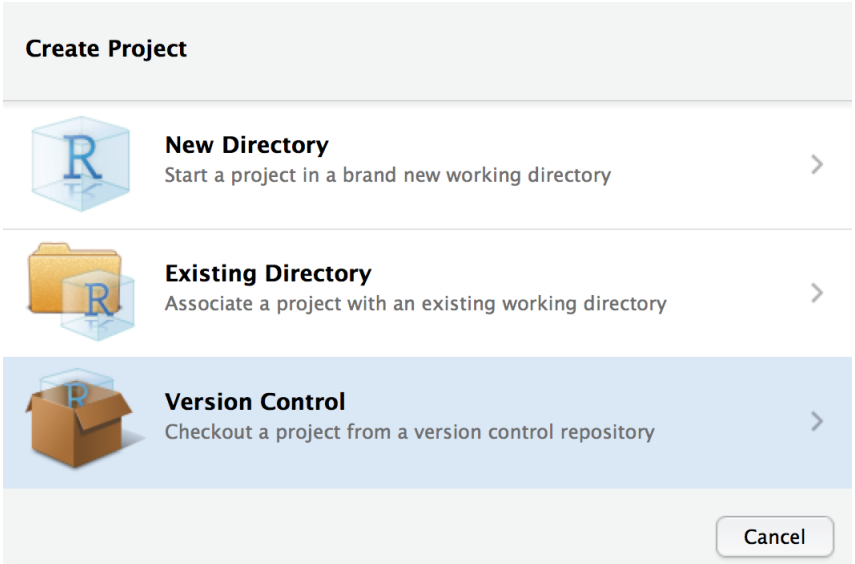

Figure B.1: VC Project Option

[1] https://cran.r-project.org/mirrors.html
[2] https://www.rstudio.com/products/rstudio/download/

(b) You will need to enter the project details:

- **Repository URL:** `https://github.com/ebeneditos/Liam`
- **Project directory name:** Name of the directory where the project will be located (it is automatically generated), namely `DIR`.
- **Create project as a subdirectory of:** Path where you want `DIR` to be located, namely `PATH`.

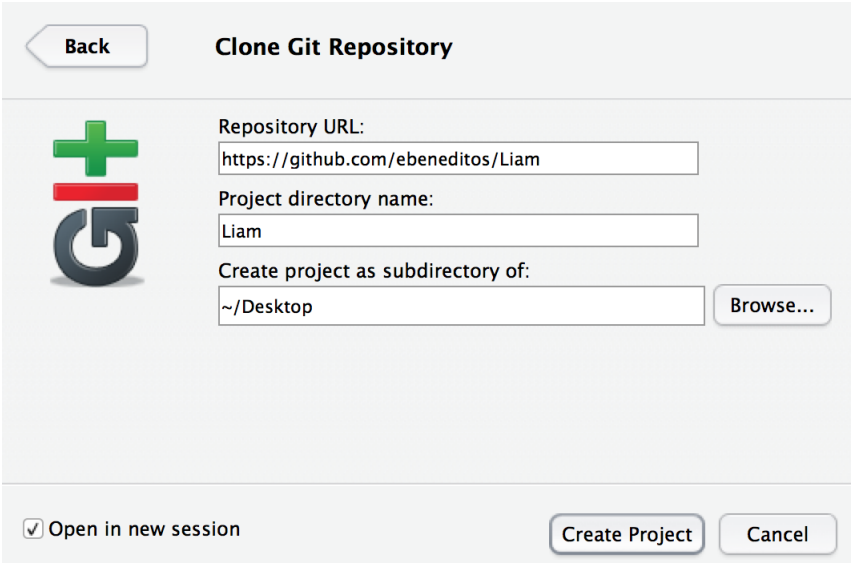

Figure B.2: Cloning Git Repository

In Figure B.2 you can find the specifications, with `DIR` name *Liam* located in desktop `PATH`.

(c) **Enter Account Credentials**. A specific GitHub account with read-only rights to the Liam Repository has been created. For privacy issues, they are not included in this report. If you are a developer, you can contact the project's author for them, his contact details are at the end of this Appendix.

3. **Set Global Variables**. There are some global variables that must be set, you can do from R terminal with the commands:

```
> user_renviron = path.expand(file.path(" ", ".Renviron"))
> file.edit(user_renviron) # Open with another text editor if this fails
```

The `.Renviron` file will appear, where you can introduce the global variables. For this project we need:

- `R_TELEGRAM_PATH_Liam`. Path where the Bot is located, specified before when cloning the Git Repository. Then, the bot path will be `PATH/DIR`.
- `R_TELEGRAM_BOT_Liam`. Bot's unique Telegram `TOKEN`. Contact the project's author for requiring it.
- `R_TELEGRAM_USER_Operator`. Telegram's *user_id* of the user that will carry the Operator role.

You can do so by adding into the `.Renviron` file the lines:

```
R_TELEGRAM_PATH_Liam=PATH/DIR

R_TELEGRAM_BOT_Liam=TOKEN

R_TELEGRAM_USER_Operator=operator_id
```

4. **Run the Bot!** Run the `Liam.R` script and the Bot will start working.

For any doubts you may have, you can contact the author of the project, Ernest Benedito, via e-mail[3], or you can check his GitHub Respository[4].

[3] `ebeneditos@gmail.com`
[4] `https://github.com/ebeneditos/TGBot`

C | Scaling the Solution

In this Appendix it is explained the process to follow in order to scale the solution to other subjects. To do so, it is needed to:

1. **Create the Dataset**. Generate a `questions.xlsx` file with the following sheets:

   (a) **Questions**. Containing the set of questions you want to train. This must have the columns:

   - **Questions Language Columns**. One column for each language available in the tool This contains one question per line translated to the multiple languages. The name of each column must be the language in lower cases (e.g. 'english').
   - **Intent**. Intent of the question.
   - **Topic**. Topic of the question.
   - **Subtopic**. Subtopic of the question.

   (b) **Keywords**. Containing information about the subject's KWs, with columns:

   - **Topic**. Topic that the KW belongs to.
   - **Subtopic**. Subtopic that the KW belongs to.
   - **Keyword**. KeyWordRoot of the KW.
   - **ID**. Unique identifier of the KW.
   - **Parents**. A hierarchy of KW should be generated, so in this column the other KWs from which the actual KW is derived, separated by commas (,).
   - **n**. Number of words of the KW.
   - **unique**. Binary variable indicating whether the KW is particular for that Topic (i.e. it does not appear in other Topics) or not.
   - **Languages**. One column for each language available in the tool. This should contain ways of saying the KW in the different languages, separated by commas (,).

2. **Update `data` directory**. As explained in the Bot's Data Structure Section 8.1.1, inside the `data` directory there are 3 files. You must update them so to fit in your solution. This can be done by:

   - **Build Models**. You will need the folder `update`, which present in the developers environment (you can contact the project's author for further information). Once there, go to the script `models.R` and run it. A `models.RData` file will be generated, containing the models for Intent, Topic and Subtopic for each language introduced.
   - **Write pre-built Answers**. You can do it for the different languages in `data`'s directory file `responses.json`. Contact the project's author for further information.
   - **Generate Database**. You can easily generate a SQLite database with the script `sqlite.R` and the table templates located located in the `sqlite/sqlite_source` directory from the `update` folder. Thus, you can generate the needed `.sqlite` file.

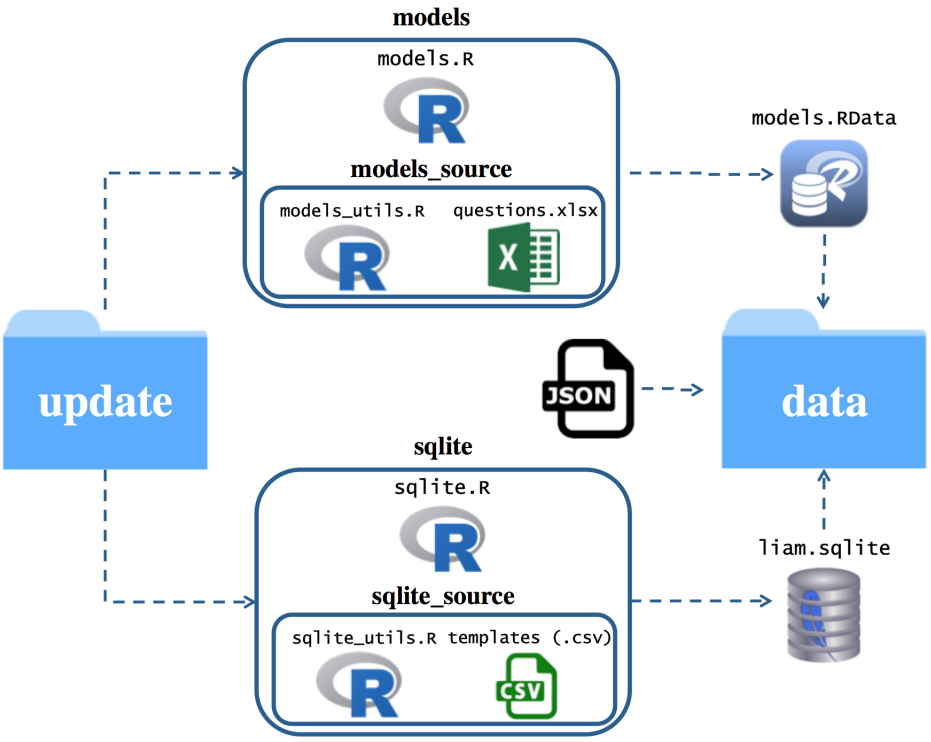

Figure C.1: Document Generation Flow

For any doubts you may have, you can contact the author of the project, Ernest Benedito, via e-mail[1], or you can check his GitHub Respository[2].

[1] `ebeneditos@gmail.com`
[2] `https://github.com/ebeneditos/TGBot`

D | Auxiliar Figures

mtry

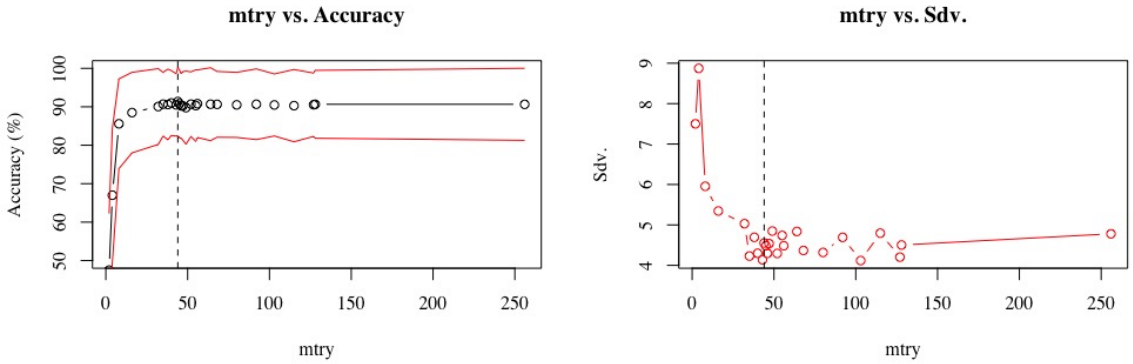

Figure D.1: Accuracy and Sdv. vs. mtry for Catalan Intent Classifier

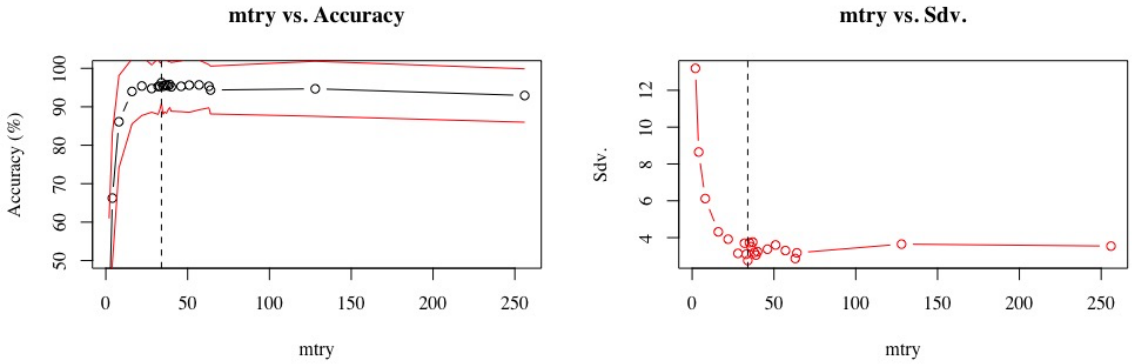

Figure D.2: Accuracy and Sdv. vs. mtry for Catalan Topic Classifier

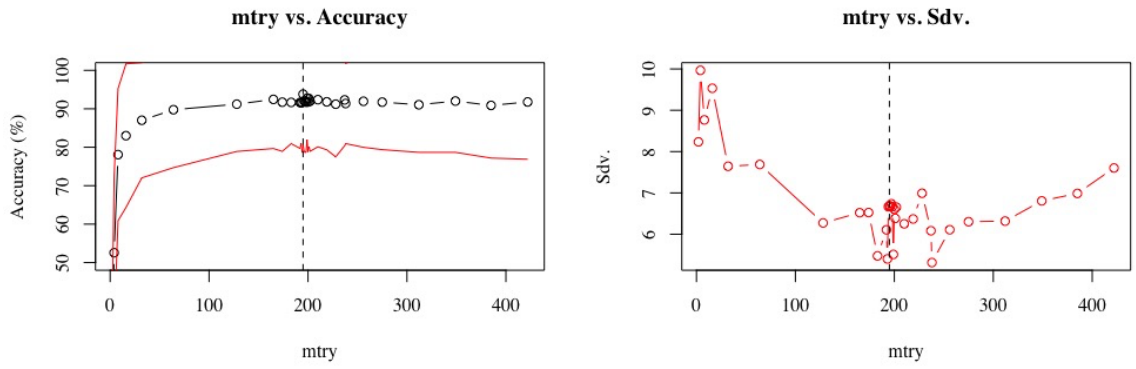

Figure D.3: Accuracy and Sdv. vs. mtry for Catalan Subtopic Classifier

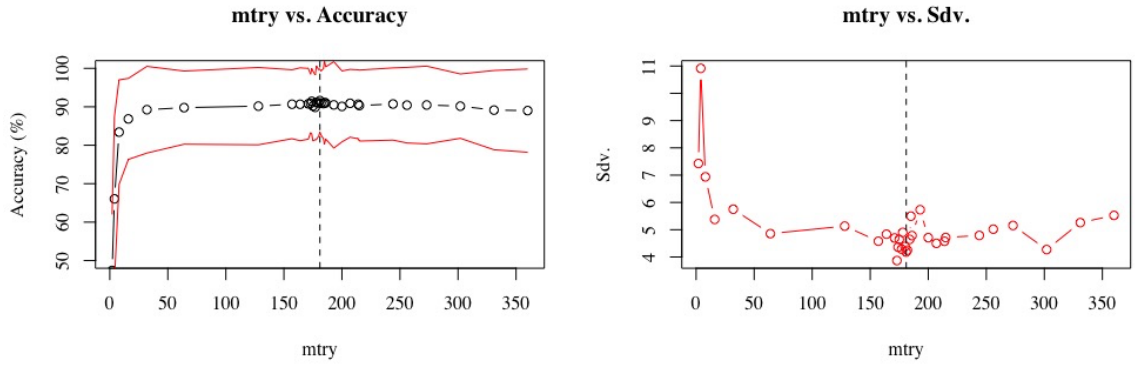

Figure D.4: Accuracy and Sdv. vs. mtry for Spanish Intent Classifier

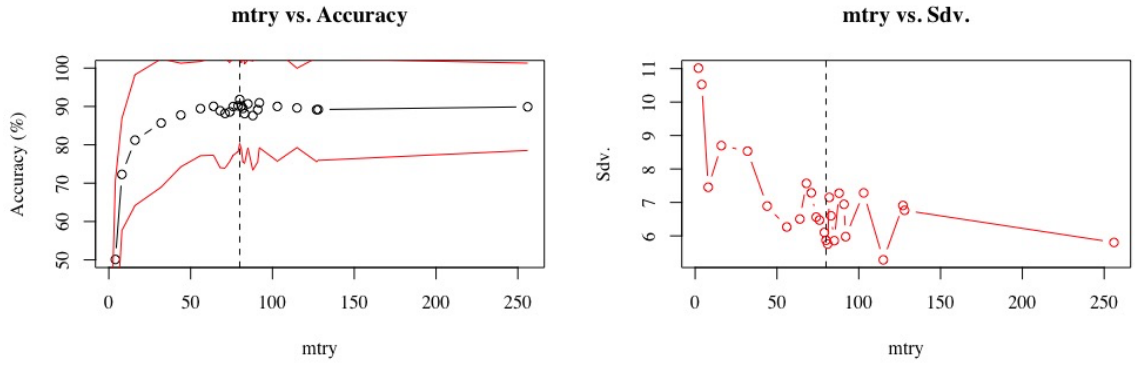

Figure D.5: Accuracy and Sdv. vs. mtry for Spanish Subtopic Classifier

ntree

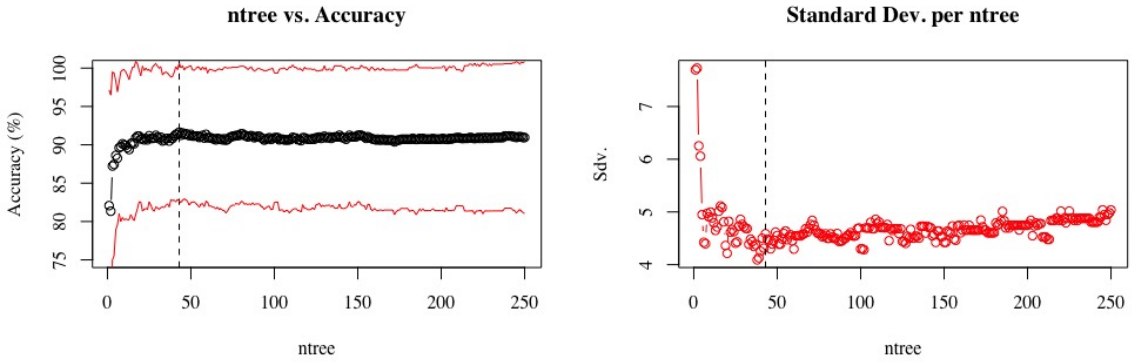

Figure D.6: Accuracy and Sdv. vs. ntree for Catalan Intent Classifier

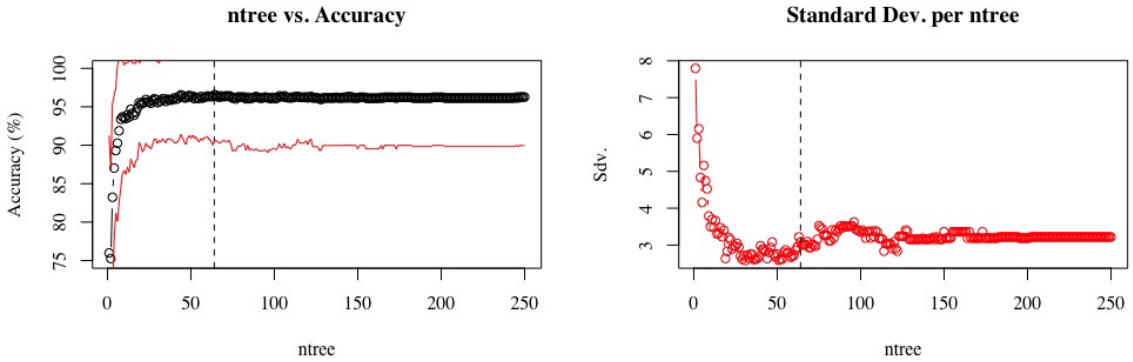

Figure D.7: Accuracy and Sdv. vs. ntree for Catalan Topic Classifier

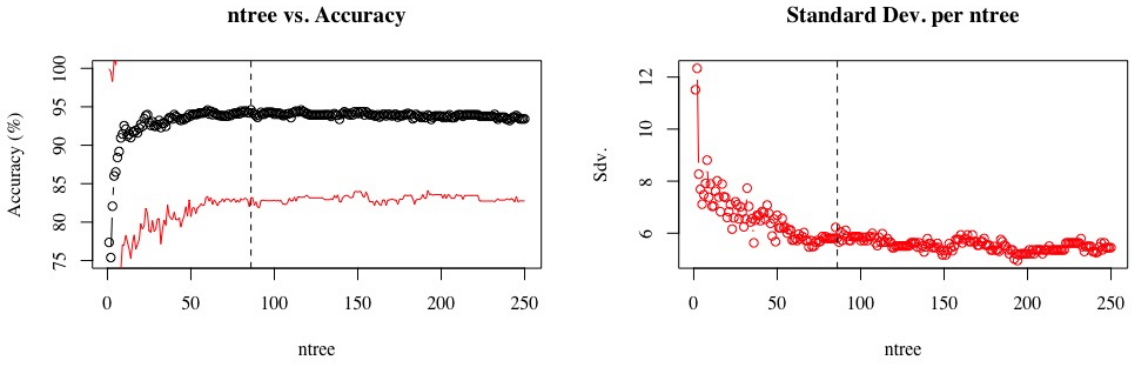

Figure D.8: Accuracy and Sdv. vs. ntree for Catalan Subtopic Classifier

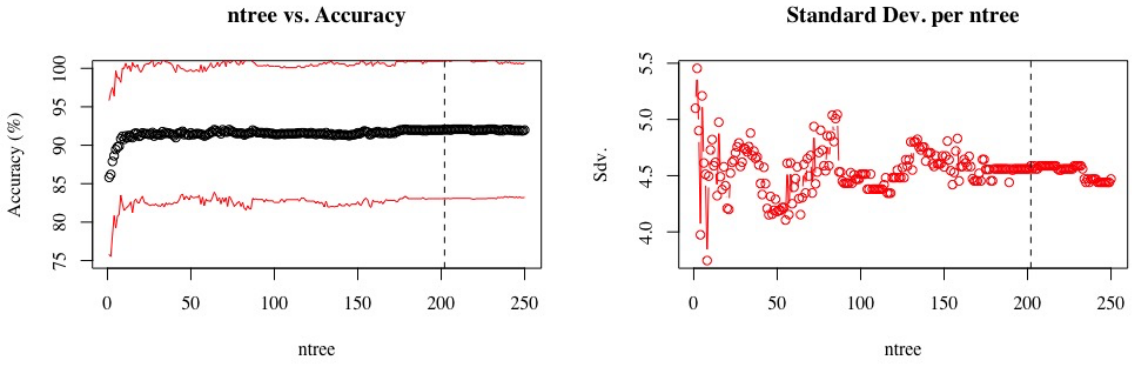

Figure D.9: Accuracy and Sdv. vs. ntree for Spanish Intent Classifier

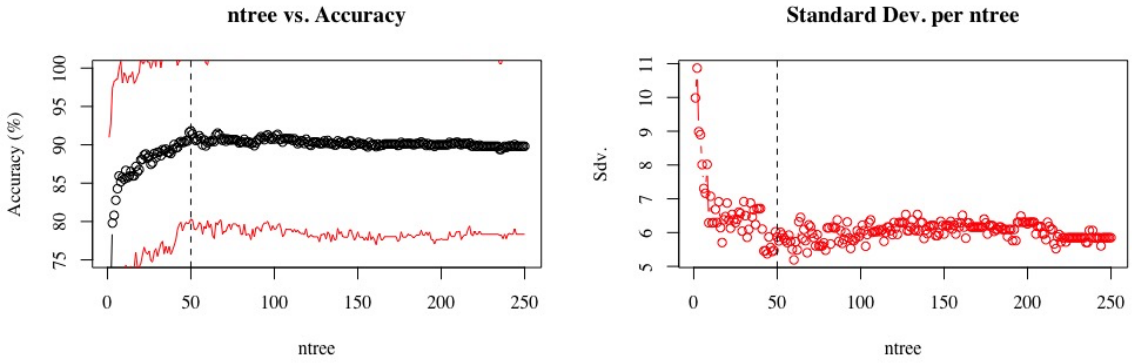

Figure D.10: Accuracy and Sdv. vs. ntree for Spanish Subtopic Classifier

Discriminant Words

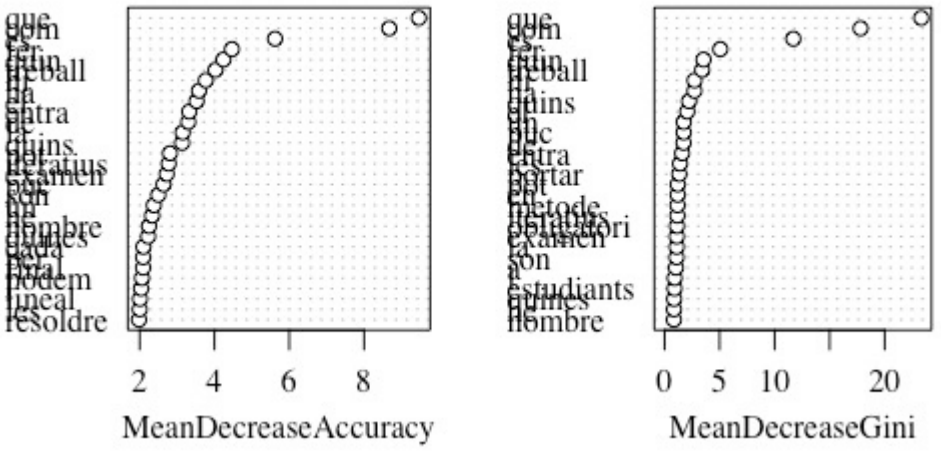

Figure D.11: Most Discriminant words for Catalan Intent Classifier

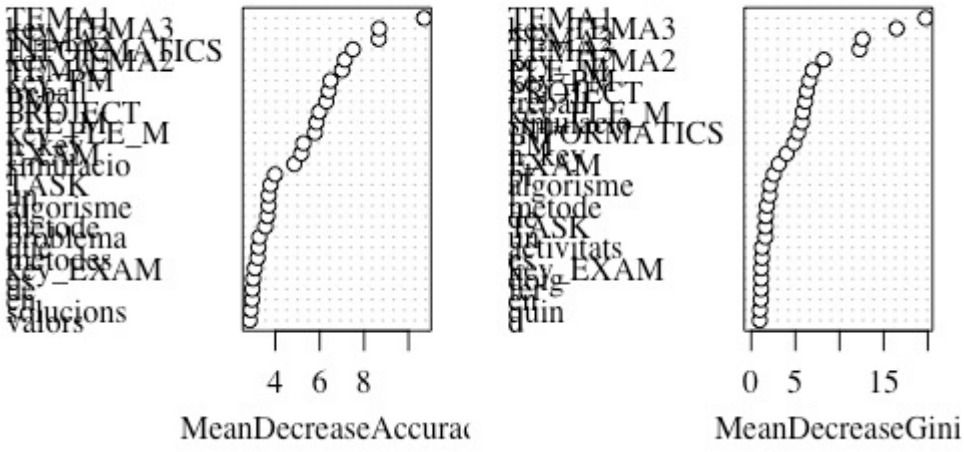

Figure D.12: Most Discriminant words for Catalan Topic Classifier

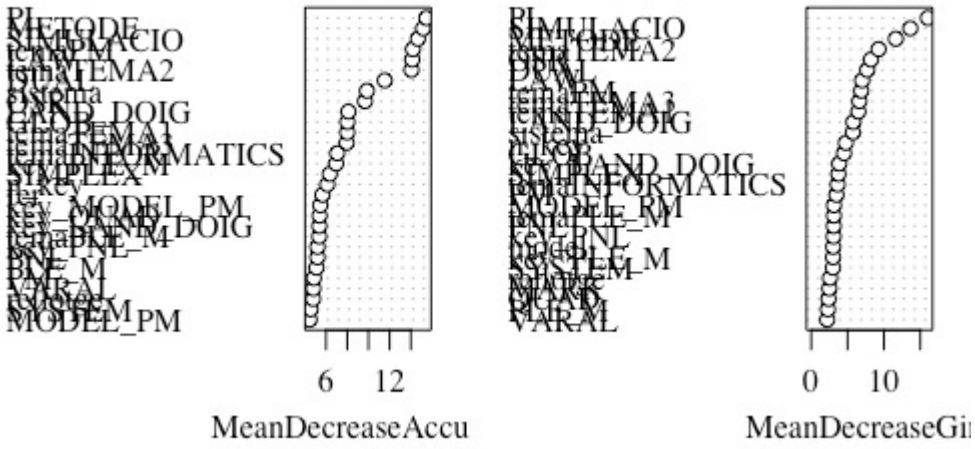

Figure D.13: Most Discriminant words for Catalan Subtopic Classifier

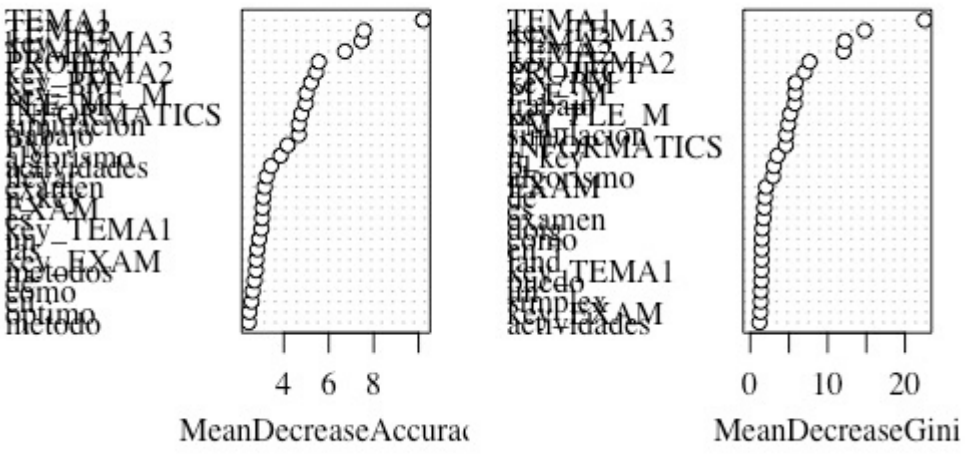

Figure D.14: Most Discriminant words for Spanish Topic Classifier

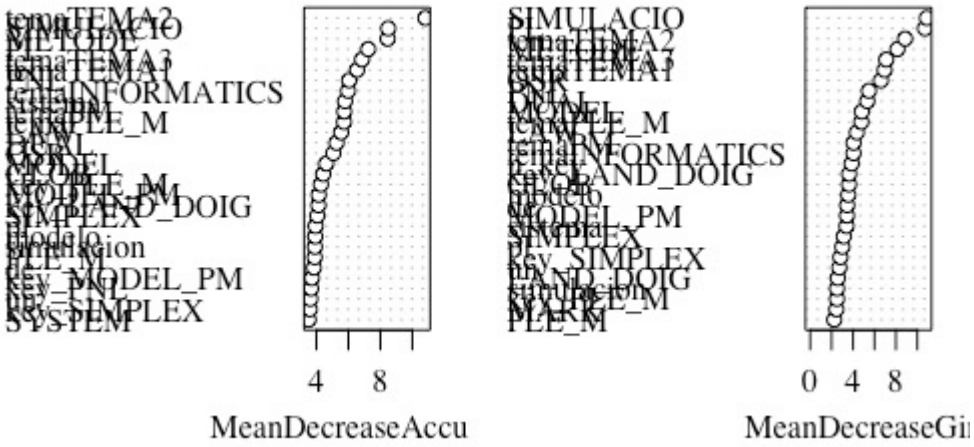

Figure D.15: Most Discriminant words for Spanish Subtopic Classifier

E | R Code

For privacy issues, only code regarding to the text processing will be attached. Code from the chatbot implementation is expected to be shared in the author's GitHub Respository[1].

E.1 ML Methods Comparison

Code used for the Machine Learning methods comparison.

E.1.1 Classifiers.R

```
library(xlsx)
library(tema)

## Set Variables -----------------------------------------------------------

y_var <- 'intent'
language <- 'catalan'

## Load Data ---------------------------------------------------------------

file <- 'models_source/questions.xlsx'

df <- read.xlsx(file)

## Generate input matrix ---------------------------------------------------

docs <- df[, language]

docs <- textCleaning(docs)

dtm <- DocumentTermMatrix(Corpus(VectorSource(docs)),
                          control = list(
                            tolower = T,
                            removePunctuation = T,
                            removeNumbers = T,
                            stripWhitespace = T,
                            wordLengths = c(1, Inf)
                          ))

dtm <- removeSparseTerms(dtm, .99)
```

[1] https://github.com/ebeneditos/TGBot

```
x <- as.data.frame(as.matrix(dtm))
if(y_var == 'subtopic') x <- cbind(x, dummy('topic', df))

dim(x)

y <- as.factor(df[, y_var])
table(y)

#### Create Data ----------------------------------------------------------------------------

# As DT
data <- copy(x)

data$status_clm <- copy(y)
table(data$status_clm)


#### Divide Sample ----------------------------------------------------------------------------

p <- 0.7
set.seed(12345)
train.sel <- sample(c(FALSE,TRUE),nrow(data),rep=TRUE,prob=c(1-p,p))
train <- data[train.sel,]
test <- data[!train.sel,]


#### Naive Bayes ----------------------------------------------------------------------------
library(e1071)

p <- c()

for (i in 1:10){
  t0<-Sys.time()
  nb <- naiveBayes(status_clm ~ ., train)
  t1<-Sys.time()
  print(time.NB <- t1 - t0)
  preds <- predict(nb, newdata = test[,-c(ncol(test))])
  (t <- table(preds, test$status_clm))
  p <- c(p, sum(diag(t))/sum(t)*100)
}

res.NB <- max(p)
cat("Accuracy:", round(res.NB, 1),"%")


#### Decision Tree ----------------------------------------------------------------------------
library(rpart)

t0<-Sys.time()
my_tree <- rpart(status_clm ~ ., train,
                 method ="class")
t1<-Sys.time()
print(time.DT <- t1 - t0)

preds <- predict(my_tree, newdata = test[,-c(ncol(test))], type = 'class')

(t <- table(preds, test$status_clm))

res.DT <- sum(diag(t))/sum(t)*100

#### Random Forest ----------------------------------------------------------------------------
library(randomForest)

par(cex=0.6)
```

```
set.seed(12345)
t.rf <- tuneRF(data[,-ncol(data)],data$status_clm, stepFactor = 2, improve = .001,
               mtryStart = 1,trace=FALSE)

t0<-Sys.time()
mtry <- t.rf[which.min(t.rf[,2]), 1]
ntree <- 200
set.seed(12345)
rf.mod <- randomForest(x=data[,-ncol(data)],y=data$status_clm,mtry=mtry,importance=TRUE,
                       ntree=ntree,do.trace=T)
t1<-Sys.time()
cat(paste('mtry:', mtry, '; ntree:', ntree))
print(time.RF <- t1 - t0)

print(rf.mod)

res.RF <- sum(diag(rf.mod$confusion))/sum(rf.mod$confusion)*100
cat("Accuracy:",round(res.RF, 1),"%")

par(cex=0.6)
plot(rf.mod)
legend("topright", colnames(rf.mod$err.rate),col=1:5,cex=0.8,fill=1:5)


#### KNN ------------------------------------------------------------------------------------
library(deldir)
library(kknn)
library(class)

p <- 0.7
n <- nrow(data)

set.seed(12345)
train.sel <- sample(c(FALSE,TRUE),n,rep=TRUE,prob=c(1-p,p))

train <- data[train.sel,]
test <- data[!train.sel,]

K <- seq(3,31,2)
p <- c()

set.seed(12345)
t0<-Sys.time()
for(k in K){
  knn <- knn(train[,-ncol(train)], test[,-ncol(test)], cl=train$status_clm, k = k)
  t <- table(knn,test$status_clm)
  p <- c(p,sum(diag(t))/sum(t))
}
t1<-Sys.time()
print(time.KNN <- t1 - t0)

plot(K,p,pch=19,type='b')
maxp <- which.max(p)
abline(v=K[maxp],lty=2)
res.KNN <- p[maxp]*100

cat("Accuracy:", round(res.KNN,1), "%")


#### Support Vector Machine -----------------------------------------------------------------

p <- 0.7
n <- dim(data)[1]
```

```
set.seed(12345)
train.sel <- sample(c(FALSE,TRUE),n,rep=TRUE,prob=c(1-p,p))

xtrain <- data.frame(data[train.sel,-ncol(data)])
xtest <- data.frame(data[!train.sel,-ncol(data)])
ytrain<-as.factor(data$status_clm[train.sel])
ytest<-as.factor(data$status_clm[!train.sel])

p <- c()

for (i in -15:3){
  t0<-Sys.time()
  model <- svm(xtrain,ytrain, gamma = 2^i)
  t1<-Sys.time()
  print(time.SVM <- t1 - t0)
  pred <- predict(model, xtest)
  p <- c(p, sum(pred==ytest)/length(ytest)*100)
}

res.SVM <- max(p)
cat("Accuracy:", round(res.SVM, 1),"%")


#### Neural Network ------------------------------------------------------------------------------
library(nnet)

#############################################################
# Divide Sampla
#############################################################
p <- 0.7               # Proporcion en muestra de entrenamiento
n <- nrow(data)        # numero de observaciones

set.seed(12345)
train.sel <- sample(c(FALSE,TRUE),n,rep=TRUE,prob=c(1-p,p))

pc <- princomp(data[,-ncol(data)])
cumsum((pc$sdev^2)/sum(pc$sdev^2)*100)

data_pc <- data[, -ncol(data)]

y.Burt <- matrix(0,ncol=length(unique(data$status_clm)),nrow=n)

for (i in 1:ncol(y.Burt)){
  y.Burt[, i] <- data$status_clm == unique(data$status_clm)[i]
}

xtrain <- data.frame(data[train.sel, -ncol(data)])
xtest <- data.frame(data[!train.sel, -ncol(data)])
ytrain<-y.Burt[train.sel,]
ytest<-y.Burt[!train.sel,]

#############################################################
# Neural Networks (4 clusters)
#############################################################

# Parameter tuning usando el test
lambdas <- seq(0, 1, by = 0.1)
n.lambdas <- length(lambdas)
lambda.v <- exp(lambdas)-1

m.min <- 1
m.max <- 5
n.m <- length(m.min:m.max)
m.v <- m.min:m.max
```

```
set.seed(1234)
prop.missclas <- matrix(data=NA, nrow=n.m,ncol=n.lambdas)
row=1
col=1
for(m in m.v){
  for(lambda in 1:n.lambdas){
    set.seed(12345)
    PC.nn <- nnet(x=xtrain, y=ytrain,
                  size=m, decay=lambda.v[lambda], softmax = TRUE, trace=F)
    hat <- apply(predict(PC.nn, newdata = xtest),
                 1,which.max)-1
    pred.tab <- table(apply(ytest, 1, which.max)-1,hat)
    prop.missclas[row,col] <- 1-sum(diag(pred.tab))/dim(xtest)[1]

    print(paste('m:', m, '; lambda:', lambda.v[lambda], '; acc:', round(100 - prop.missclas[row,col]*100,
         1)))

    col=col+1
  }
  row=row+1
  col=1
}

best.lambda <- lambda.v[which(prop.missclas==min(prop.missclas),arr.ind=T)[1,2]]
best.m <- m.min - 1 +which(prop.missclas==min(prop.missclas),arr.ind=T)[1,1]

t0<-Sys.time()
set.seed(12345)
PC.nn <- nnet(x=xtrain, y=ytrain,
              size=best.m, decay=best.lambda, softmax = TRUE, trace=F)
t1<-Sys.time()
print(time.NN <- t1 - t0)

hat <- apply(predict(PC.nn, newdata = xtest),
             1,which.max)-1
(pred.tab <- table(apply(ytest, 1, which.max)-1,hat))

res.NN<- sum(diag(pred.tab))/sum(pred.tab)*100
cat("Accuracy:", round(res.NN, 1), "%")

#### Model Comparison ------------------------------------------------------------------------

m <- matrix(data = NA, nrow = 6, ncol = 2)
rownames(m) <- c("NB", "KNN", "SVM", "DT", "RF", "NN")
colnames(m) <- c("Accuracy (%)", "Training Time (s)")
m[,1]<- round(c(res.NB, res.KNN, res.SVM, res.DT, res.RF, res.NN), 1)
m[,2]<- round(as.numeric(c(time.NB, time.KNN, time.SVM, time.DT, time.RF, time.NN), units = 'secs'), 1)
m
```

E.2 Tune the RF Parameters

Code used for the tuning of the Random Forest classifiers parameters.

E.2.1 Models.R

```
library(data.table)
library(tm)
library(dummies)
library(openxlsx)
library(stringi)

source('ModelsUtils.R')


## Set Variables ---------------------------------------------------------

y_var <- 'intent'
language <- 'spanish'

## Load Data -------------------------------------------------------------

x <- getInput(docs, df, y_var = y_var, KeyWords$main)

dim(x)

y <- as.factor(df[, y_var])
table(y)


## Parameter Tuning ------------------------------------------------------

folds <- data_part(n = nrow(x), times = 20, p = .9)

best.mtry <- optim_mtry(x, y, folds, mtries = NULL, ntree = 50, iter = NULL, sum = sum <- NULL)

best.ntree <- optim_ntree(x, y, folds, ntree.max = 250, best.mtry)


## Final Model -----------------------------------------------------------

t0<-Sys.time()
set.seed(12345)
rf.mod <- randomForest(x = x, y = y, mtry = best.mtry, importance = TRUE,
                       ntree = best.ntree, do.trace = T)
t1<-Sys.time()

time.RF <- difftime(t1, t0, units = "secs")

varImpPlot(rf.mod)


## Performance Analysis --------------------------------------------------

y_pred <- matrix(ncol = length(unique(y)), nrow = nrow(x))
colnames(y_pred) <- c(levels(y))
questions <- data.frame(question = df[, language], y = y, pred = NA, prob = NA)

for (i in 1:nrow(x)){
  print(paste('Checking Question', i, 'of', nrow(x)))
```

```
  set.seed(12345)
  rf.modi <- randomForest(x = x[-i,], y = y[-i], mtry = best.mtry, ntree = best.ntree)
  pred <- as.numeric(predict(object = rf.modi, newdata = x[i,], type = 'prob'))*100
  y_pred[i, ] <- c(pred)
  questions$pred[i] <- levels(y)[which.max(pred)]
  questions$prob[i] <- max(pred)
}

questions <- cbind(questions, x_aux)

table(questions$pred, questions$y)
res.RF <- round(mean(questions$pred == questions$y)*100, 1)

miss.quest <- questions[questions$pred != questions$y, ]
miss.quest <- miss.quest[with(miss.quest, order(-prob)),]
write.xlsx(miss.quest, paste0('missmatch_questions_', y_var, '.xlsx'))


## Percentile accuracy analysis ------------------------------------------------

ths <- seq(0, 100, by <- 5)
y_acc <- c()

p_acc <- c()
p_qty <- c()

for (th in ths){
  percentile <- questions$prob >= th & questions$prob < th + by
  p_qty <- c(p_qty, sum(percentile))
  p_acc <- c(p_acc, ifelse(sum(percentile) > 0, mean(questions$pred[percentile] ==
       questions$y[percentile])*100, 0))
}

plot(ths, p_qty, main = 'Amount of Claims by Percentile', xlab = 'Percentile', ylab = '# Claims', type =
     'h')
plot(ths, p_acc, main = 'Accuracy by Percentile', xlab = 'Percentile', ylab = 'Accuracy', type = 'b', xlim
     = c(0, 100), ylim = c(0, 100))

perc_out <- cumsum(p_qty)/sum(p_qty)*100
perc_acc <-
     cumsum((p_qty*p_acc)[length(p_qty):1])[length(p_qty):1]/cumsum(p_qty[length(p_qty):1])[length(p_qty):1]
perc_improve <- perc_acc - min(perc_acc)

plot(ths, perc_improve, type = 'l', main = 'Accuracy improving by threshold', ylab = '% Improve')
lines(ths, perc_out, col = 2, type = 'l')

plot(ths, diff_vec(perc_improve)/by, type = 'l', main = 'Derivate improving', ylab = 'Diff. Improve')
lines(ths, diff_vec(perc_out)/by, col = 2, type = 'l')

abline(v = th <- 50, lty = 2, col = 2)


## Summary ---------------------------------------------------------------------

cat('\nModel Details:\n',
    'mtry:', best.mtry, '\n',
    'ntree:', best.ntree, '\n',
    'Training time:', format(.POSIXct(time.RF,tz="GMT"), "%Hh %Mmin %Ss"), '\n',
    'Accuracy:', res.RF, '%\n\n',
    'Proposed minimum threshold:', th/100, '\n',
    'Questions under threshold:', round(perc_out[ths == th], 1), '%\n',
    'Accuracy remaining questions:', round(perc_acc[ths == th], 1), '%\n\n'
    )
```

E.2.2 ModelsUtils.R

```
#### TRAINING MODEL FUNCTIONS ####

getInput <- function(docs, df, y_var = 'intent', kw){

  x <- as.data.frame(getDTM(docs))

  if(y_var == 'subtema') x <- cbind(x, dummy('tema', df))

  if(y_var %in% c('tema', 'subtema')){

    kw_lan <- strsplit(kw[, language], ', ')

    x_aux <- data.frame(n_key = rep(0,nrow(x)))

    for (tema in unique(kw[, y_var])){
      x_aux[, tema] <- 0
      x_aux[, paste0('key_', tema)] <- 0
    }

    for (l in 1:nrow(x)){
      n_words <- x[l, x[l,] > 0]
      words <- rep(names(n_words), n_words)
      for (i in 1:nrow(kw)){
        i_words <- kw_lan[[i]]
        weigth <- sum(i_words %in%words)/kw$n[i]
        if (weigth >= 1){weigth <- 1 + 0.1*kw$n[i]}
        if (weigth > 0){
          x_aux[l, kw[, y_var][i]] <- x_aux[l, kw[, y_var][i]] + weigth

          if (weigth >= 1){
            x_aux[l, 'n_key'] <- x_aux[l, 'n_key'] + 1
          }

          if (weigth >= 1 & kw$key[i] == 1){
            x_aux[l, paste0('key_', kw[, y_var][i])] <- x_aux[l, paste0('key_', kw[, y_var][i])] + weigth
          }
        }
      }
    }
    x <- cbind(x, x_aux)
  }

  return(x)
}

trainRF <- function(x, df, y_var, language, p = 0.8){

  if (y_var == 'subtema') p <- 0.9

  t0<-Sys.time()

  ## Generate input matrix ------------------------------------------------------

  y <- as.factor(df[, y_var])

  ## Parameter Tuning -----------------------------------------------------------

  folds <- data_part(n = nrow(x), times = 20, p = p)

  best.mtry <- optim_mtry(x, y, folds, mtries = NULL, ntree = 50, iter = NULL, sum = sum <- NULL)
```

```
  best.ntree <- optim_ntree(x, y, folds, ntree.max = 250, best.mtry)


  ## Final Model ----------------------------------------------------------------

  set.seed(12345)
  rf.mod <- randomForest(x = x, y = y, mtry = best.mtry, importance = TRUE,
                         ntree = best.ntree, do.trace = T)

  res.RF <- round(rf.mod$err.rate[rf.mod$ntree, 1]*100, 1)

  print(rf.mod$confusion)

  varImpPlot(rf.mod)

  t1<-Sys.time()
  time.RF <- difftime(t1, t0, units = "secs")

  cat('\nModel Details:\n',
      'mtry:', best.mtry, '\n',
      'ntree:', best.ntree, '\n',
      'Training time:', format(.POSIXct(time.RF,tz="GMT"), "%Hh %Mmin %Ss"), '\n',
      'OOB:', res.RF, '%\n\n'
  )

  return(rf.mod)
}


#### PARAMETER OPTIMIZATION FUNCTIONS ####

data_part <- function(n, times = 1, p = 0.7){

  folds <- matrix(ncol = times, nrow = n)

  for (i in 1:times){
    set.seed(i)
    folds[, i] <- sample(c(TRUE, FALSE), n, rep = TRUE, prob = c(p, 1 - p))
  }

  return(folds)
}

pow2 <- function(from, to){
  pow2vals <- c()
  i <- 1
  while (T){
    nextval <- 2^i
    if (nextval <= to){
      pow2vals <- c(pow2vals, nextval)
      i <- i + 1
    }
    else{
      return(pow2vals)
    }
  }
}

optim_mtry <- function(x, y, folds, mtries = NULL, ntree = 100, iter = 1, it = 1, sum = NULL){

  cross <- ncol(folds)

  if (is.null(mtries)){
```

```
  # mtries <- fib(from = max(1, round(sqrt(ncol(x))/3)), to = ncol(x))
  mtries <- pow2(from = 1, to = ncol(x))
}

len.mtries <- length(mtries)

cat('\nIteration', it, '\n')
cat('mtries chosen:', mtries, '\n')

acc <- c()
if(is.null(sum)){sum <- data.frame()}

# ntrees <- c()

for (mtry in mtries) {
  err <- c()
  for (i in 1:cross){
    set.seed(i)
    train.sel <- folds[, i]
    test.sel <- !(folds[, i])
    rf.mod <- randomForest(x = x[train.sel,], y = y[train.sel], xtest = x[test.sel,], ytest =
        y[test.sel],
                           mtry = mtry, importance = TRUE, ntree = ntree, do.trace = F)
    err <- c(err, 100 - rf.mod$test$err.rate[ntree, 1]*100)
  }

  # n.a <- 100 - rf.mod$test$err.rate[, 1]*100
  # b.ntree <- which.max(n.a)
  # ntrees <- c(ntrees, b.ntree)
  # res.RF <- n.a[b.ntree]

  res.RF <- mean(err)
  sdv <- sd(err)

  acc <- c(acc, res.RF)

  plot(mtries[1:length(acc)], acc, xlab = 'mtry', ylab = 'Accuracy (%)', type = 'b', xlim =
      c(min(mtries), max(mtries)))

  cat('mtry:', mtry, '; Accuracy (%):', round(res.RF, 1), '\n')

  sum <- rbind(sum, c(mtry, res.RF, sdv))
  names(sum) <- c('mtry', 'acc', 'sdv')
}

sum <- sum[with(sum, order(mtry)),]

m.a <- sum$acc
best.mtry <- sum$mtry[which.max(m.a)]
best.sdv <- sum$sdv[which.max(m.a)]

cat('\nBest mtry:', best.mtry, '\n')
cat('Std. Dev.:', best.sdv, '\n')
cat('Accuracy (%):', round(max(sum$acc), 1), '\n')

if (!is.null(iter)){
  if (it == iter){
    return(best.mtry)
  }
}
else{

  min.mtry <- ifelse(which.max(m.a) == 1, 1, sum$mtry[which.max(m.a) - 1])
  max.mtry <- ifelse(which.max(m.a) == length(m.a), ncol(x), sum$mtry[which.max(m.a) + 1])
```

```
plot(sum$mtry, sum$acc, main = 'mtry vs. Accuracy', xlab = 'mtry', ylab = 'Accuracy (%)', type = 'b',
     xlim = c(min(min.mtry, min(mtries)), max(max.mtry, max(mtries))))

if (min.mtry == 1 & max.mtry == ncol(x)){
  if(cross>1){
    plot(sum$mtry, sum$sdv, col = 2, main = 'mtry vs. Sdv.', xlab = 'mtry', ylab = 'Sdv.', type = 'b')
    abline(v=best.mtry,lty=2)

    plot(sum$mtry, sum$acc, main = 'mtry vs. Accuracy', xlab = 'mtry', ylab = 'Accuracy (%)', type =
         'b', ylim = c(50, 100))
    lines(sum$mtry, sum$acc + 1.96*sum$sdv, type = 'l', col = 2)
    lines(sum$mtry, sum$acc - 1.96*sum$sdv, type = 'l', col = 2)
    abline(v=best.mtry,lty=2)
  }

  else{
    plot(sum$mtry, sum$acc, main = 'mtry vs. Accuracy', xlab = 'mtry', ylab = 'Accuracy (%)', type =
         'b', ylim = c(50, 100))
    abline(v=best.mtry,lty=2)
  }

  return(best.mtry)
}

len <- length(mtries)
l <- 1:len

mtries <- min.mtry:max.mtry
mtries <- mtries[!(mtries %in% sum$mtry)]

if(length(mtries) > len){
  mtries <- mtries[round(l*length(mtries)/len)]
}
if (length(mtries) == 0){

  if(cross>1){
    plot(sum$mtry, sum$sdv, col = 2, main = 'mtry vs. Sdv.', xlab = 'mtry', ylab = 'Sdv.', type = 'b')
    abline(v=best.mtry,lty=2)

    plot(sum$mtry, sum$acc, main = 'mtry vs. Accuracy', xlab = 'mtry', ylab = 'Accuracy (%)', type =
         'b', ylim = c(50, 100))
    lines(sum$mtry, sum$acc + 1.96*sum$sdv, type = 'l', col = 2)
    lines(sum$mtry, sum$acc - 1.96*sum$sdv, type = 'l', col = 2)
    abline(v=best.mtry,lty=2)
  }

  else{
    plot(sum$mtry, sum$acc, main = 'mtry vs. Accuracy', xlab = 'mtry', ylab = 'Accuracy (%)', type =
         'b', ylim = c(50, 100))
    abline(v=best.mtry,lty=2)
  }

  return(best.mtry)
}
else{

  abline(v = best.mtry, lty = 2)
  abline(v = min.mtry, col = 2, lty = 2)
  abline(v = max.mtry, col = 2, lty = 2)

  return(optim_mtry(x = x, y = y, folds = folds, mtries = mtries, ntree = ntree, iter = iter, it =
       it+1, sum = sum))
}
```

```
  }
}

optim_ntree <- function(x, y, folds, ntree.max = 200, best.mtry){

  cross <- ncol(folds)

  mat <- matrix(ncol = cross, nrow = ntree.max)
  times <- c()

  for (i in 1:cross){
    cat('Building Forest', i, 'of', cross, '\n')
    set.seed(i)
    train.sel <- folds[, i]
    test.sel <- !(folds[, i])

    t0<-Sys.time()
    rf.mod <- randomForest(x = x[train.sel,], y = y[train.sel], xtest = x[test.sel,], ytest = y[test.sel],
                           mtry = best.mtry, importance = TRUE, ntree = ntree.max)
    t1<-Sys.time()

    mat[, i] <- 100 - rf.mod$test$err.rate[, 1]*100

    times <- c(times, time.RF <- difftime(t1, t0, units = "secs"))
  }

  time.RF <- mean(times)

  n.a <- rowMeans(mat)
  best.ntree <- which.max(n.a)
  res.RF <- n.a[best.ntree]

  sdv <- apply(mat, 1, sd)

  if(cross > 1){
    plot(1:ntree.max, sdv, col = 2, main = 'Standard Dev. per ntree', xlab = 'ntree', ylab = 'Sdv.', type
         = 'b')
    abline(v = best.ntree, lty = 2)
  }

  plot(1:ntree.max, n.a, main = 'ntree vs. Accuracy', xlab = 'ntree', ylab = 'Accuracy (%)', type = 'b',
       ylim = c(75, 100))
  abline(v = best.ntree, lty = 2)

  if(cross > 1){
    lines(1:ntree.max, n.a - 1.96*sdv, type = 'l', col = 2)
    lines(1:ntree.max, n.a + 1.96*sdv, type = 'l', col = 2)
  }

  cat('\nBest ntree:', best.ntree, '\n')
  cat('Accuracy (%):', round(res.RF, 1), '\n')
  cat('Mean Training Time:', format(.POSIXct(time.RF,tz="GMT"), "%Hh %Mmin %Ss"))

  return(best.ntree)
}
```

Index

Bibliography

[1] Chatbot. (2017, December 21). *Chatbot — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/Chatbot

[2] CogniApps. (2016, December 27). *Historia de los Chatbots.* https://medium.com/@cogniapps/historia-de-los-chatbots-bd71f3fd914a

[3] Turing test. (2018, January 4). *Turing test — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/Turing_test

[4] ELIZA. (2017, December 3). *ELIZA — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/ELIZA

[5] PARRY. (2017, December 10). *PARRY — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/PARRY

[6] Jabberwacky. (2017, December 19). *Jabberwacky — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/Jabberwacky

[7] Watson (computer). (2018, January 4). *Watson (computer) — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/Watson_(computer)

[8] Wakefield, J. (2016, March 24). *Microsoft chatbot is taught to swear on Twitter.* http://www.bbc.com/news/technology-35890188

[9] Hyken, S. (2017, July 15). *AI And Chatbots Are Transforming The Customer Experience.* https://www.forbes.com/sites/shephyken/2017/07/15/ai-and-chatbots-are-transforming-the-customer-experience/#3e50ce0841f7

[10] Nakpodia, E. (2017, January 30). *Chatbots and The Future of Education.* https://chatbotsmagazine.com/chatbots-and-the-future-of-education-6c08db8f4c62

[11] Letzter, R. (2016, May 15). *IBM's brilliant AI just helped teach a grad-level college course.* http://uk.businessinsider.com/watson-ai-became-a-teaching-assistant-2016-5

[12] McNeal, M. (2016, December 7). *A Siri for Higher Ed Aims to Boost Student Engagement.* https://www.edsurge.com/news/2016-12-07-a-siri-for-higher-ed-aims-to-boost-student-engagement

[13] Hubert. (2017, May 30). *6 Ways Artificial Intelligence and Chatbots Are Changing Education.* https://chatbotsmagazine.com/six-ways-a-i-and-chatbots-are-changing-education-c22e2d319bbf

[14] Méndez, J. R., Cid, I., Glez-Peña, D., Rocha, M., and Fdez-Riverola, F. (2008). *A comparative impact study of attribute selection techniques on naive bayes spam filters.* Industrial Conference on Data Mining (Springer), 213–227.

[15] Seewald, A. K. (2007). *An evaluation of naive bayes variants in content-based learning for spam filtering.* Intelligent Data Analysis (11 May), 497–524.

[16] Kosmopoulos, A., Paliouras, G., and Androutsopoulos, I. (2008). *Adaptive spam filtering using only naive bayes text classifiers.* Proceedings of the Fifth Conference on Email and Anti-Spam (Citeseer).

[17] Schneider, K.-M. (2003). *A comparison of event models for naive bayes anti-spam e- mail filtering.* Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics. Vol. 1, 307–314.

[18] Go, A., Bhayani, R., and Huang, L. (2009). *Twitter sentiment classification using distant supervision.* CS224N Project Report, Stanford, 1:12.

[19] Zhang, C., Zeng, D., Li, J., Wang, F.-Y., and Zuo, W. (2009). *Sentiment analysis of chinese documents: From sentence to document level.* Journal of the American Society for Information Science and Technology, 2474–2487.

[20] Boiy, E., Hens, P., Deschacht, K., and Moens, M.-F. (2007). *Automatic sentiment analysis in on-line text.* ELPUB, pages 349–360.

[21] Dai, W., Xue, G.-R., Yang, Q., and Yu, Y. (2007). *Transferring naive bayes classifiers for text classification.* Proceedings of the national conference on artificial intelligence, Vol. 22, 540.

[22] McCallum, A., Nigam, K., et al. (1998). *A comparison of event models for naive bayes text classification.* AAAI-98 workshop on learning for text categorization (Citeseer), Vol. 752, 41–48.

[23] Yong, Z., Youwen, L., and Shixiong, X. (2009, March). *An Improved KNN Text Classification Algorithm Based on Clustering.* Journal of Computers, Vol. 4, No. 3.

[24] Al-Shalabi, R., and Obeidat, R. (2008, March). *Improving KNN Arabic text classification with n-grams based document indexing*. Proceedings of the Sixth International Conference on Informatics and Systems, Cairo, Egypt, 108-112

[25] Rogati, M., and Yang, Y. (2002, November). *High-performing feature selection for text classification*. Proceedings of the eleventh international conference on Information and knowledge management (ACM), 659-661.

[26] Soucy, P., and Mineau, G. W. (2001). *A simple KNN algorithm for text categorization*. Data Mining. ICDM, Proceedings IEEE International Conference, 647-648.

[27] OpenCV 2.4.13.5 documentation (2018, January 3). *Introduction to Support Vector Machines*. `https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`

[28] Zhang, D., and Lee, W. S. (2003). *Question classification using support vector machines*. Proceedings of the 26th annual international ACM SIGIR conference on Re- search and development in informaion retrieval, 26–32.

[29] Leopold, E., and Kindermann, J. (2002). *Text categorization with support vector machines. Wow to represent texts in input space?*. Machine Learning, 46(1-3):423– 444.

[30] Tong, S., and Koller, D. (2001). *Support vector machine active learning with applications to text classification*. Journal of machine learning research, 2(Nov):45–66.

[31] Joachims, T. (2001). *A statistical learning learning model of text classification for support vector machines*. Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, 128–136.

[32] Joachims, T. (1998). *Text categorization with support vector machines: Learning with many relevant features*. European conference on machine learning (Springer), 137– 142.

[33] Drucker, H., Wu, D., and Vapnik, V. N. (1999). *Support vector machines for spam categorization*. IEEE Transactions on Neural networks, 10(5):1048–1054.

[34] Aggarwal, C. C., and Zhai, C. (2012). *A survey of text classification algorithms*. Mining text data (Springer), 163–222.

[35] Apte, C., Damerau, F. J., and Weiss, S. M. (2001). *Method for improvement accuracy of decision tree based text categorization*. US Patent 6,253,169 (26 June).

[36] Lewis, D. D., and Ringuette, M. (1994). *A comparison of two learning algorithms for text categorization*. Third annual symposium on document analysis and informa- tion retrieval, Vol. 33, 81–93.

[37] Wu, Q., Ye, Y., Zhang, H., Ng, M. K., and Ho, S.-S. (2014). *Forestexter: an efficient random forest algorithm for imbalanced text categorization*. Knowledge-Based Systems, 67:105–116.

[38] Jurka, T. P., Collingwood, L., Boydstun, A. E., Grossman, E., and van Atteveldt, W. (2013). *Rtexttools: A supervised learning package for text classification*. The R Journal, 5(1):6–12.

[39] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). *Natural language processing (almost) from scratch*. Journal of Machine Learning Research, 12(Aug), 2493-2537.

[40] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). *A neural probabilistic language model*. Journal of machine learning research, 3(Feb), 1137-1155.

[41] Collobert, R., and Weston, J. (2008, July). *A unified architecture for natural language processing: Deep neural networks with multitask learning*. Proceedings of the 25th international conference on Machine learning, 160-167.

[42] Document-term matrix. (2015, December 23). *Document-term matrix — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/wiki/Document-term_matrix`

[43] Brownlee, J. (2016, January 15). *Use R For Machine Learning*. `https://machinelearningmastery.com/use-r-for-machine-learning/`

[44] Nielsen, J. (2009). *Usability Engineering*. Morgan Kaufmann.

[45] Card, S. K., Robertson, G. G., and Mackinlay, J. D. (1991). *The information visualizer: An information workspace*. Proc. ACM CHI'91 Conf. (New Orleans, LA, 28 April-2 May), 181-188.

[46] Miller, R. B. (1968). *Response time in man-computer conversational transactions*. Proc. AFIPS Fall Joint Computer Conference Vol. 33, 267-277.

[47] Myers, B. A. (1985). *The importance of percent-done progress indicators for computer-human interfaces*. Proc. ACM CHI'85 Conf. (San Francisco, CA, 14-18 April), 11-17.

[48] Miteva, S. (2017, August 21). *5 Programming Languages You Can Use to Create Chatbots*. `http://valosohub.com/blog/2017/08/21/programming-languages-chatbots/`

[49] Willems, K. (2015, May 12). *Choosing R or Python for Data Analysis? An Infographic*. `https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis`

[50] Kopf, D. (2017, September 22). *If you want to upgrade your data analysis skills, which programming language should you learn?*. `https://qz.com/1063071/the-great-r-versus-python-for-data-science-debate/`

[51] Lobo, J. (2017, August 16). *How to choose the best channel for your chatbot.* https://www.inbenta.com/en/blog/chatbot-choose-best-channel-chatbot/

[52] Briz, V. (2017, March 29). *The 3 Best Platforms for your ChatBot.* https://chatbotslife.com/the-3-best-platforms-for-your-chatbot-d2693289950d

[53] Bearon, J. (2016, December 2). *How to choose the best channel for your chatbot.* https://recast.ai/blog/how-to-choose-the-best-channel-for-your-bot-the-ultimate-cheat-sheet/

[54] Brisson, K. (2016, August 25). *11 best messaging platforms for your chatbot or conversational app.* https://blog.init.ai/pick-your-platform-wisely-c5ab5bc7555d

[55] Kadam, V. (2017, October 24). *How do I build a simple Whatsapp bot?.* https://www.quora.com/How-do-I-build-a-simple-Whatsapp-bot

[56] Kahn, J. (2016, July 26). *WhatsApp announces over 1 billion daily active users & other stats.* https://9to5mac.com/2017/07/26/whatsapp-one-billion-active-daily-users/

[57] Schwartz, J. (2016, May 24). *The Most Popular Messaging App in Every Country.* https://www.similarweb.com/blog/worldwide-messaging-apps

[58] WhattApp Blog. (2017, September 5). *Building for People, and Now Businesses.* https://blog.whatsapp.com/10000633/Building-for-People-and-Now-Businesses

[59] Dogtiev, A. (2017, December 5). *Telegram Revenue and Usage Statistics.* http://www.businessofapps.com/data/telegram-statistics/

[60] Telegram (messaging service). (2018, January 1). *Telegram (messaging service) — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/Telegram_(messaging_service)

[61] Karush–Kuhn–Tucker conditions. (2017, December 1). *Karush–Kuhn–Tucker conditions — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/Karush-Kuhn-Tucker_conditions

[62] Dwyer, G. (2016, November 10). *Building a Chatbot using Telegram and Python (Part 1).* https://www.codementor.io/garethdwyer/building-a-telegram-bot-using-python-part-1-goi5fncay

[63] Push technology. (2018, January 2). *Push technology — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/Push_technology#Long_polling

[64] Braglia, L. (2016, September 17). *telegram: R Wrapper Around the Telegram Bot API.* https://cran.r-project.org/package=telegram

[65] RStudio. (Accessed: 2018, January 3). *SQLite — Databases using R — RStudio.* http://db.rstudio.com/databases/sqlite/

[66] R-core. (Accessed: 2018, January 3). *try function — R Documentation.* https://www.rdocumentation.org/packages/base/versions/3.4.3/topics/try

[67] Universitat Politècnica de Catalunya — BarcelonaTECH. (Accessed: 2018, January 3). *The Institute — Institut d'Organització i Control de Sistemes Industrials.* https://ioc.upc.edu/en/the-institute